T3000 Building Automation System

Get Started with GitHub

Temco Controls

Jay

Table of Contents

1. Introduction to Git
2. Installing Tools
   1. Installation of Git
   2. Installation of Github for Windows
   3. Procedire to Merge using Git Bash
3. Compilation on Microsoft Visual Studio 2010
4. Git commands summary
5. Cloning a repo from github.com using Git Bash Shell
6. TortoiseGit
   1. Installation of msysgit, the windows version of Git
   2. Installation of TortoiseGit
   3. Sign up for a github account
   4. Setting up SSH key
   5. Working with repository
7. TortoiseGit andGithub workflow for T3000\_Building\_Automation\_System project

# Introduction to Git

While developing software where developers are spread across the globe, issues of version management become critical. What if two developers simultaneously work on the same module and try to upload their changes back? Situations like these are not uncommon in real-world. To solve problems such as these there exist Version Control Systems.

## What is Git?

Git is a distributed version control system (dvcs) written in the programming language C. A distributed version control system keeps track of software revisions and allows many developers to work on a given project without necessarily being connected to a common network.

There is no central server which stores the data. Every local copy contains full history of source code.

Git keeps track of all versions. Therefore, you can revert to any point in your source code history.

The logo of git looks like below:



Git is a [distributed revision control](http://en.wikipedia.org/wiki/Distributed_revision_control) and [source code management](http://en.wikipedia.org/wiki/Source_code_management) (SCM) system with an emphasis on speed. Git was initially designed and developed by [Linus Torvalds](http://en.wikipedia.org/wiki/Linus_Torvalds) for [Linux kernel](http://en.wikipedia.org/wiki/Linux_kernel) development; it has since been adopted by many other projects. Every Git [working directory](http://en.wikipedia.org/wiki/Working_directory) is a full-fledged [repository](http://en.wikipedia.org/wiki/Repository_(version_control)) with complete history and full revision tracking capabilities, not dependent on network access or a central server. Git is [free software](http://en.wikipedia.org/wiki/Free_software) distributed under the terms of the [GNU General Public License](http://en.wikipedia.org/wiki/GNU_General_Public_License) version 2.

## Local and Remote Repositories

In a distributed version control system everyone has a complete copy of the source code (including the complete history of the source code) and can perform version control operations against this local copy. The use of a dvcs does not require a central code repository.

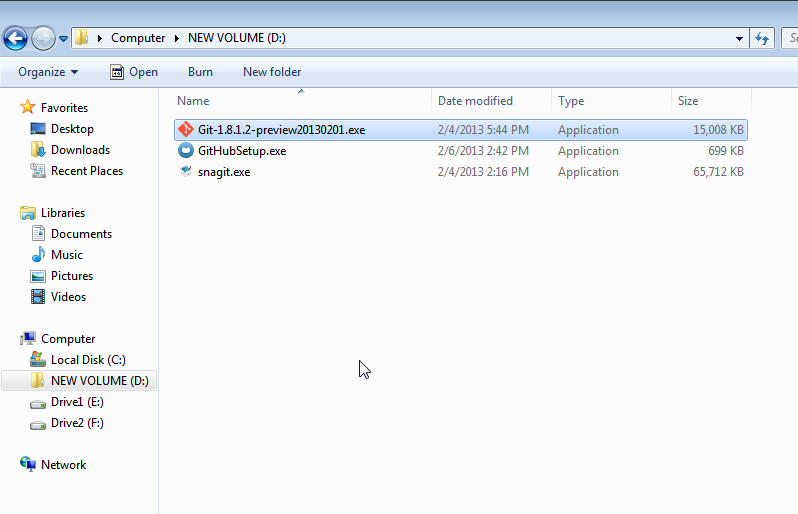
Git commits file changes to your local repository and you can synchronize your repository with other (remote) repositories. Git allows you to clone repositories, e.g. create an exact copy of a repository including the complete history of the source code. Owners of repositories can synchronize changes via push (transferring changes to a remote repository) or pull (getting changes from a remote repository).

Refer to <http://git-scm.com/book> for detailed help on understanding git.

# Installing Tools

# Installation of git

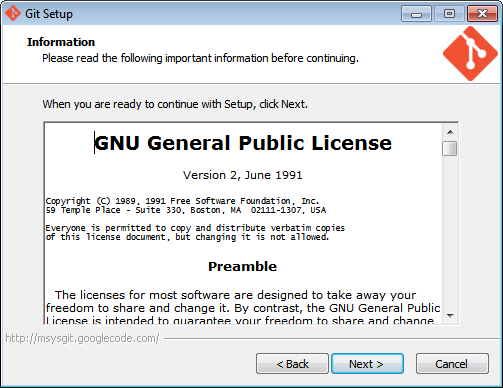
1. Download git installer.
2. Double click on the setup file.



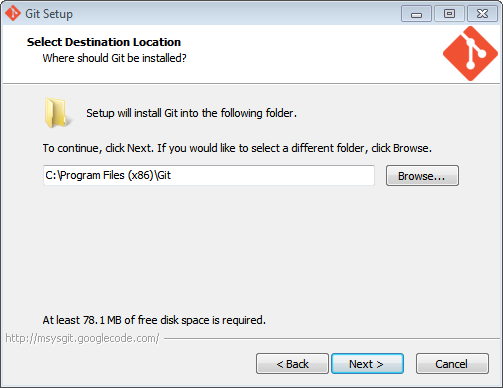
1. Following dialog box opens. Click Next to continue.



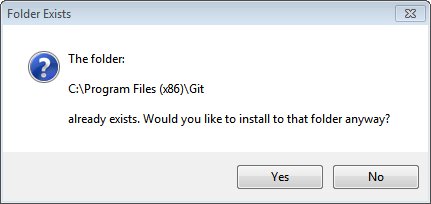
1. Read GNU General Public License. Click Next.



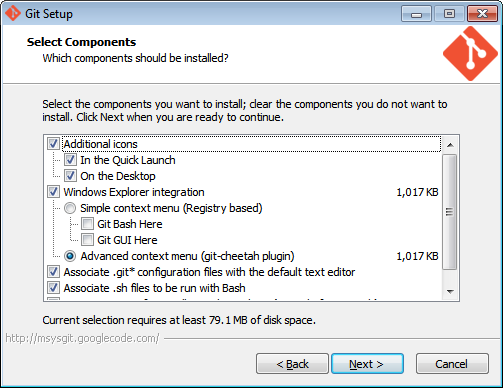
1. Select destination location. This is the directory where you want to install git. Normally, default directory can be kept as it is. If you wish, you can change the directory by clicking on Browse button. Here, we will install git in the default directory. (C:\Program Files (x86)\Git). Click Next.



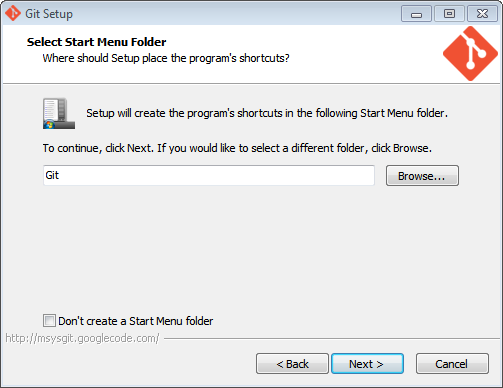
1. If following dialog box appears, click Yes.



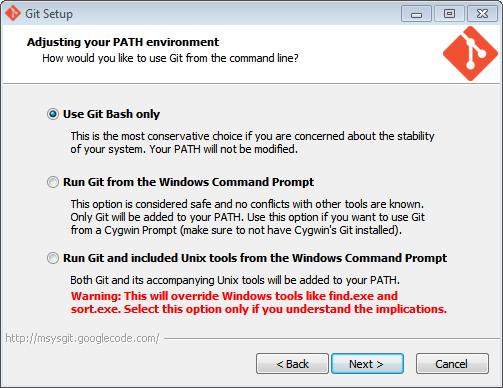
1. Following screen appears. Leave all the default options as is and click Next.



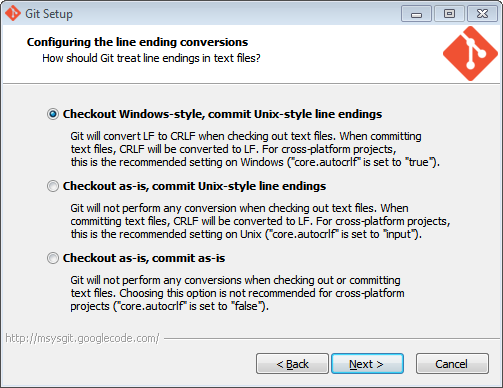
1. Following screen appears. Click Next.



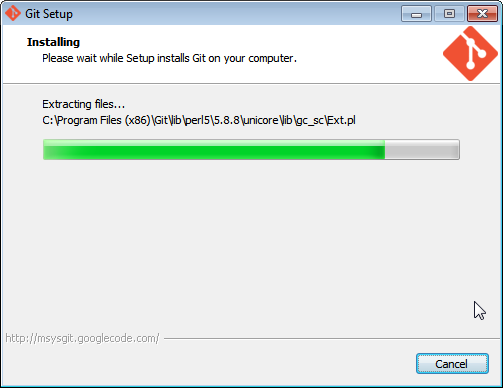
1. Following screen appears. Click Next to continue.



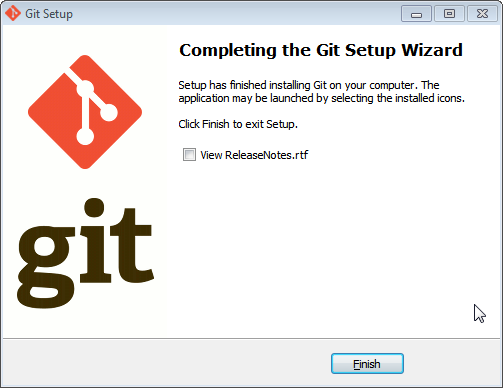
1. Following screen appears. Click Next to continue.



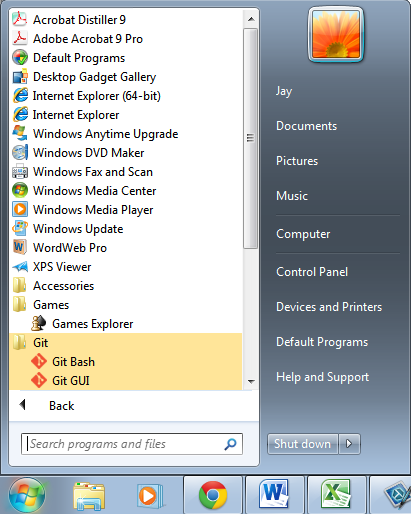
1. Setup will progress.



1. Click Finish to complete the setup.



1. Check installation in Start Menu.

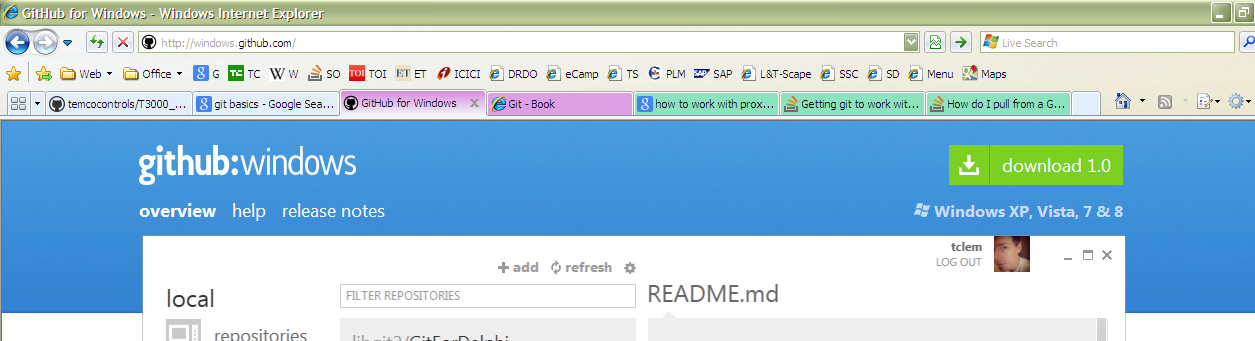


As can be seen, there are two sub menus under Git menu.

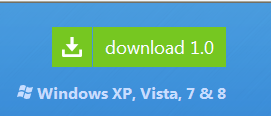
1. Git Bash
2. Git GUI

# Installation of github for windows

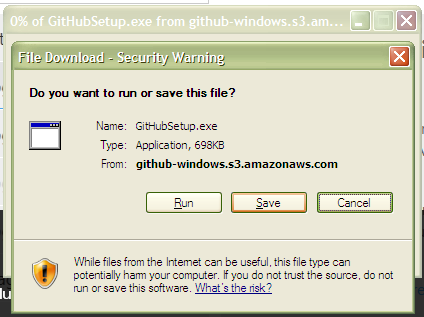
1. Download github for windows from http://windows.github.com/



1. Click on the link given for download.

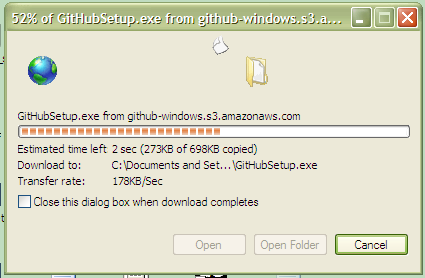


1. If any confirmation is asked by OS, click Save.



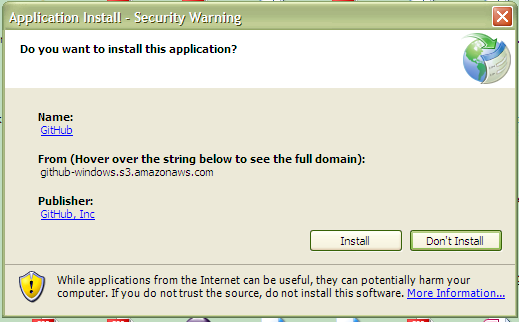
Save the file on your computer at a chosen location.

1. File download progress bar would be shown.

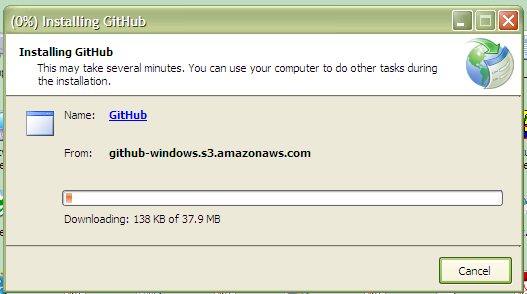


Once the download is complete, double-click on the same to install github for windows on your PC.

1. A dialog box for Security warning may appear.

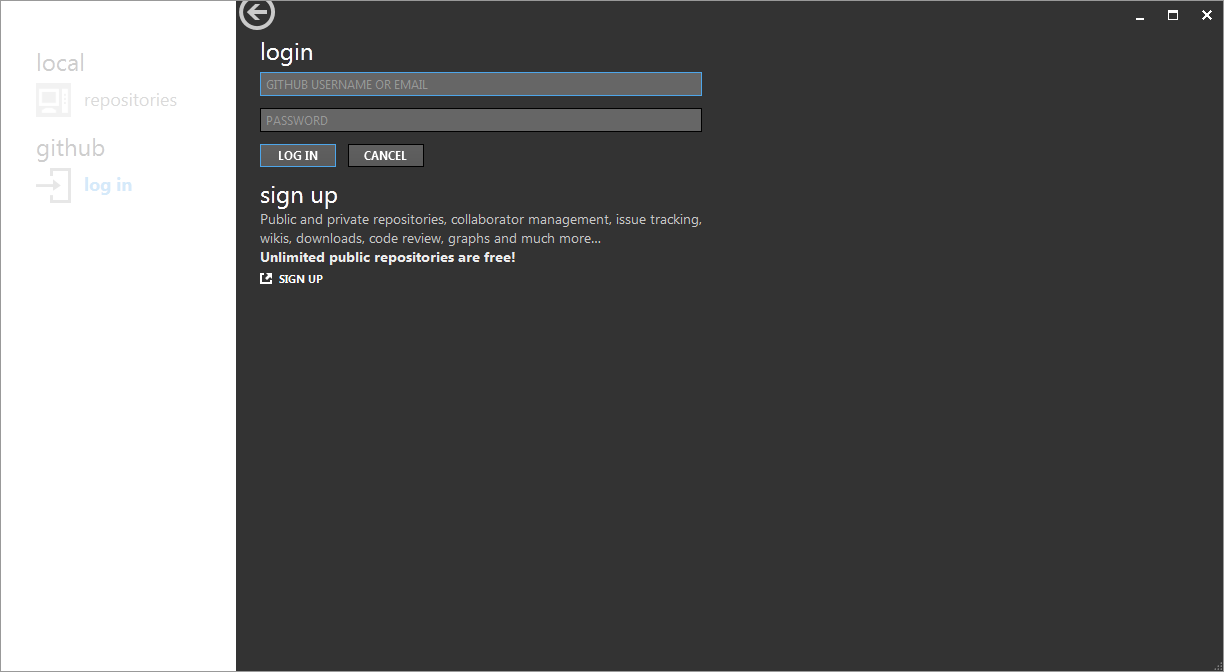


1. Github for windows starts installing.

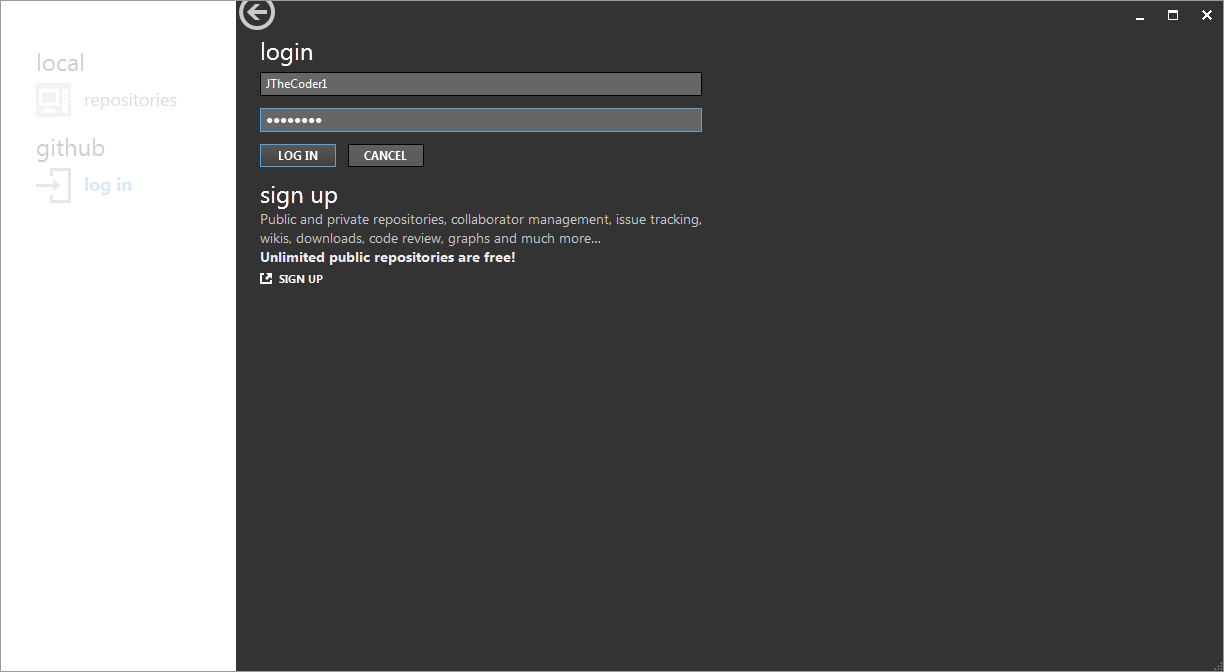


Istallation eventually gets over. Application starts.

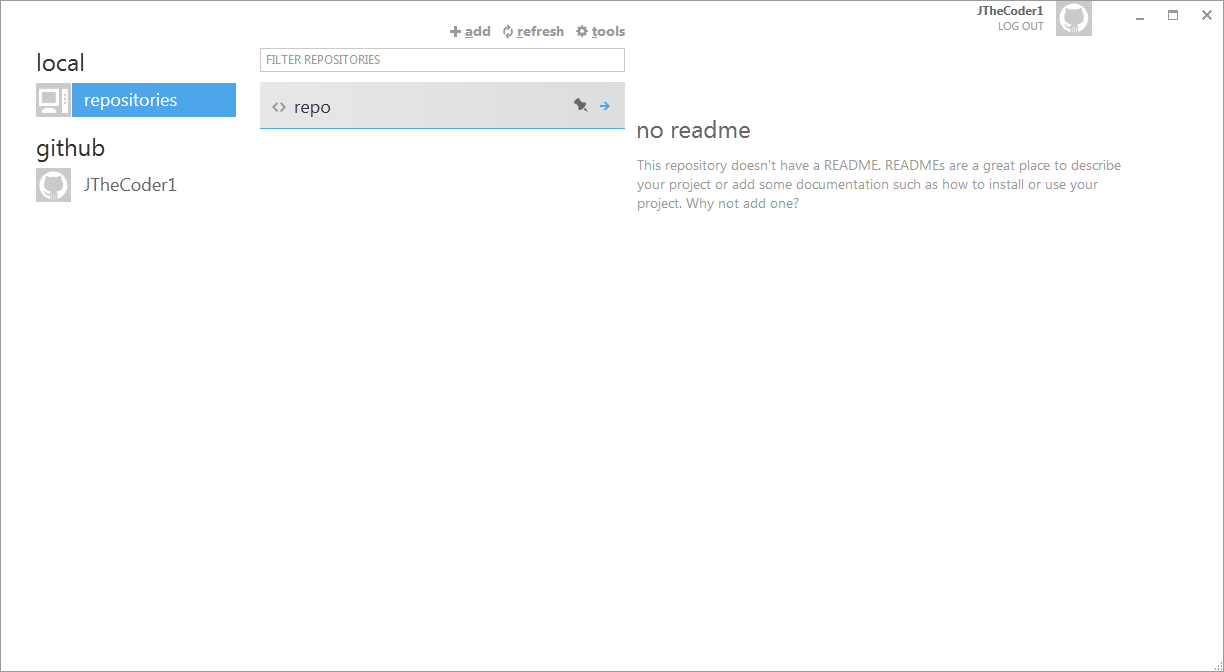
1. The first screen in the application Github for Windows is as follows.



1. Enter username and password in the given fields. You have to Sign Up with github to have these information.

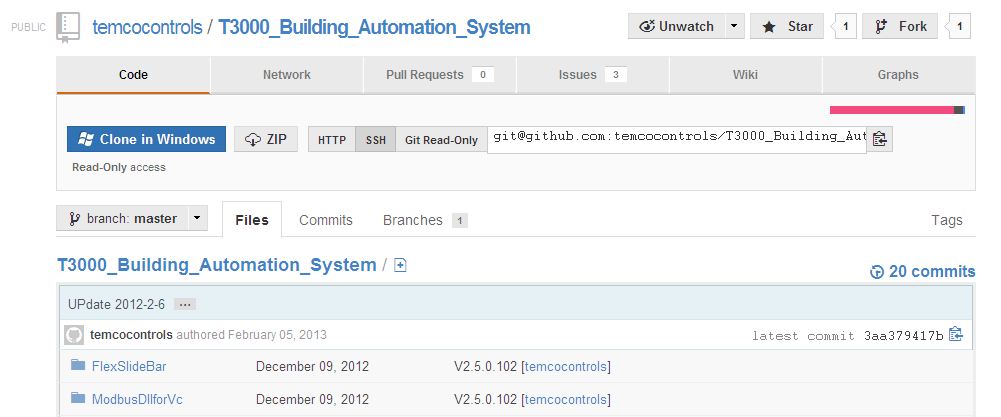


1. Application screen looks as follows.

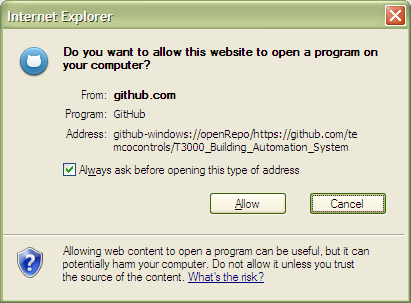


1. Once the application is installed on the PC, go to github website. Open the project you want to work on. For example, in my case, I am opening repository of T3000 Building Automation System.

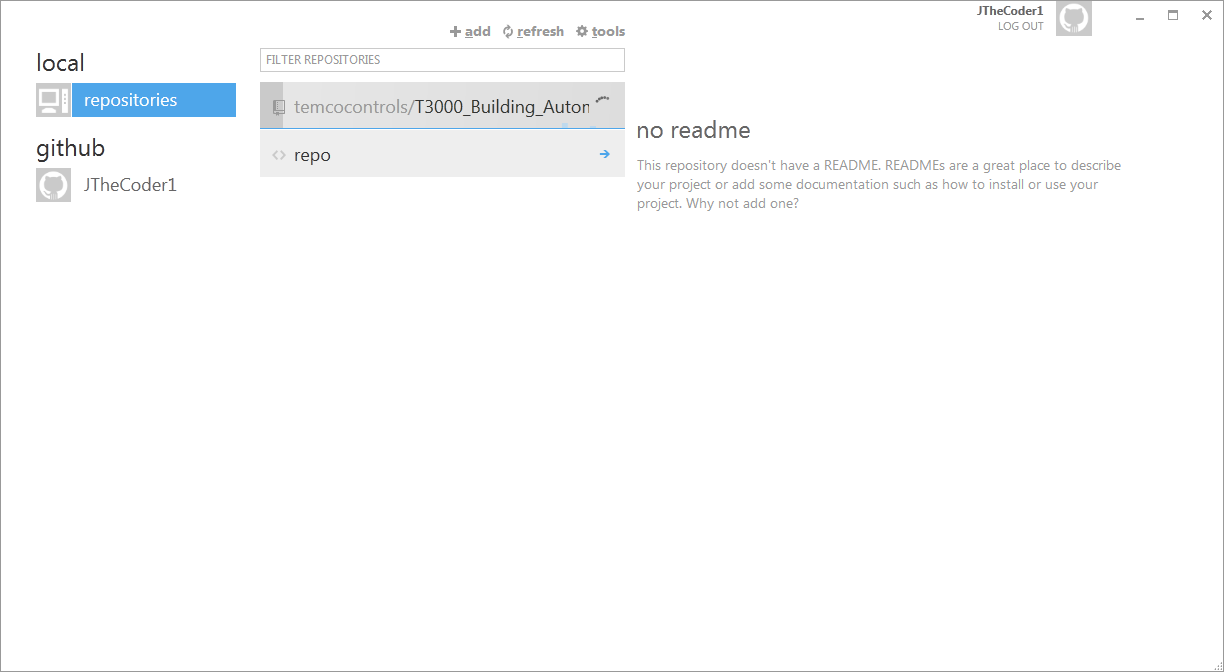
In order to clone the repository from github to local PC, click on **Clone in Windows** button as shown below.



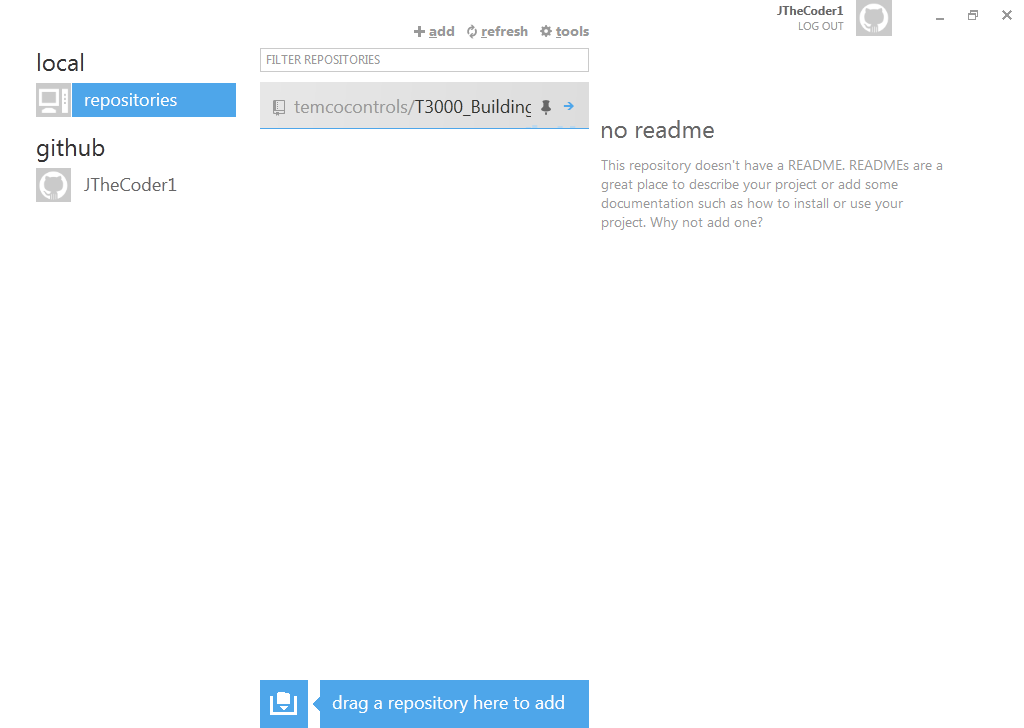
1. You may get following warning message. Click Allow.



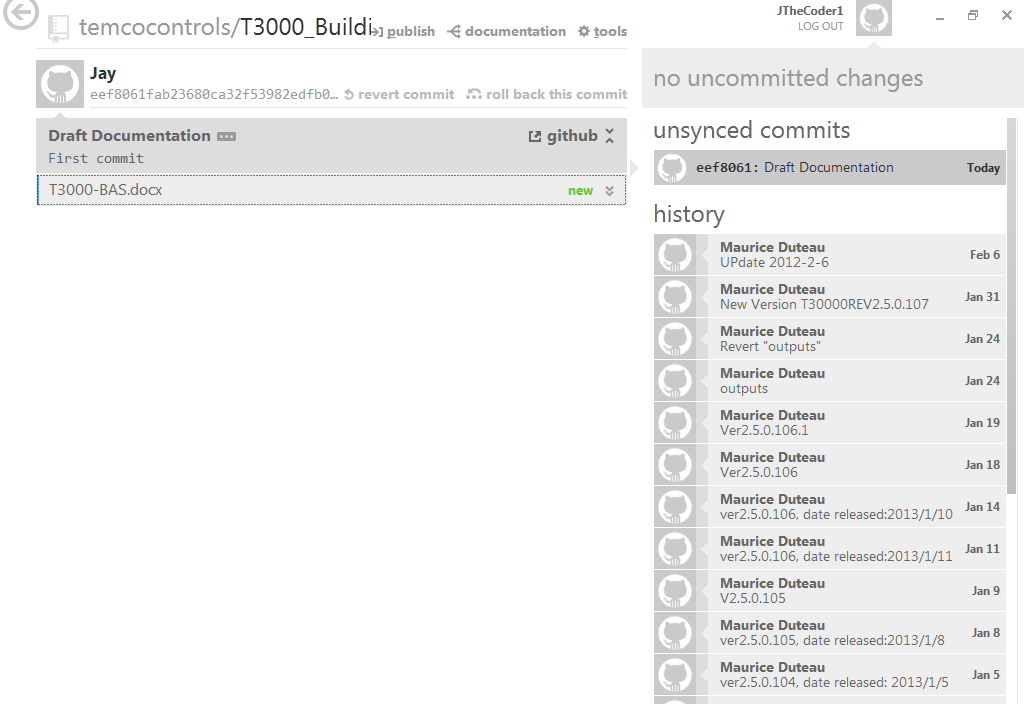
1. The repository starts getting cloned on local PC. The application shows the progress of the same. Be patient. The repository, depending on its size, takes few minutes to download.



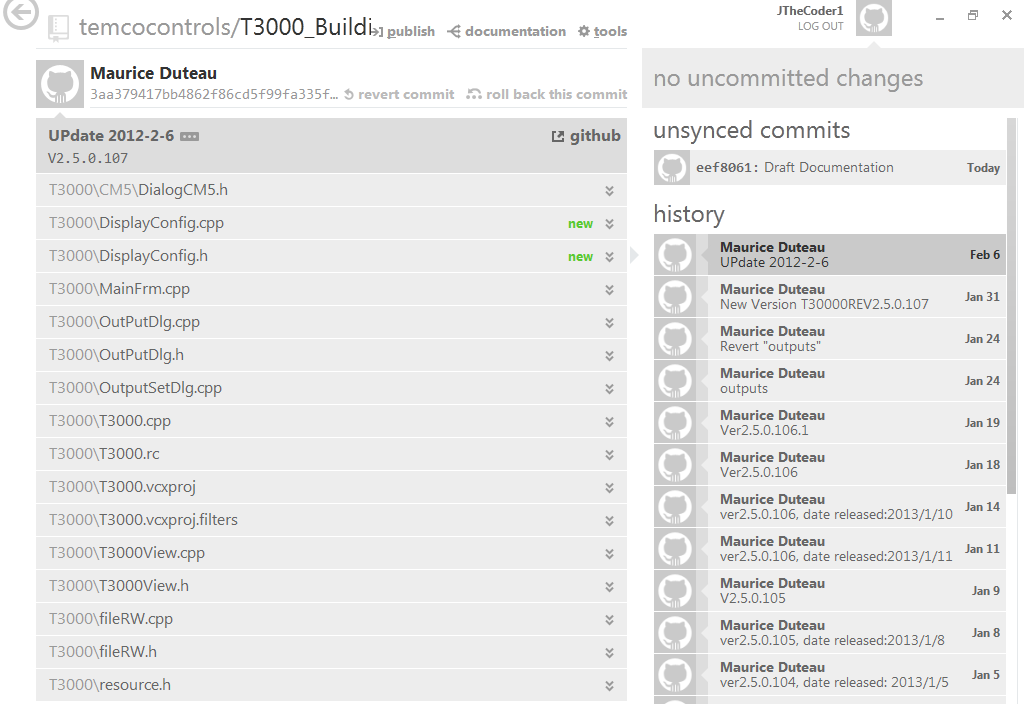
1. Once cloning is over, the repositories look like this.



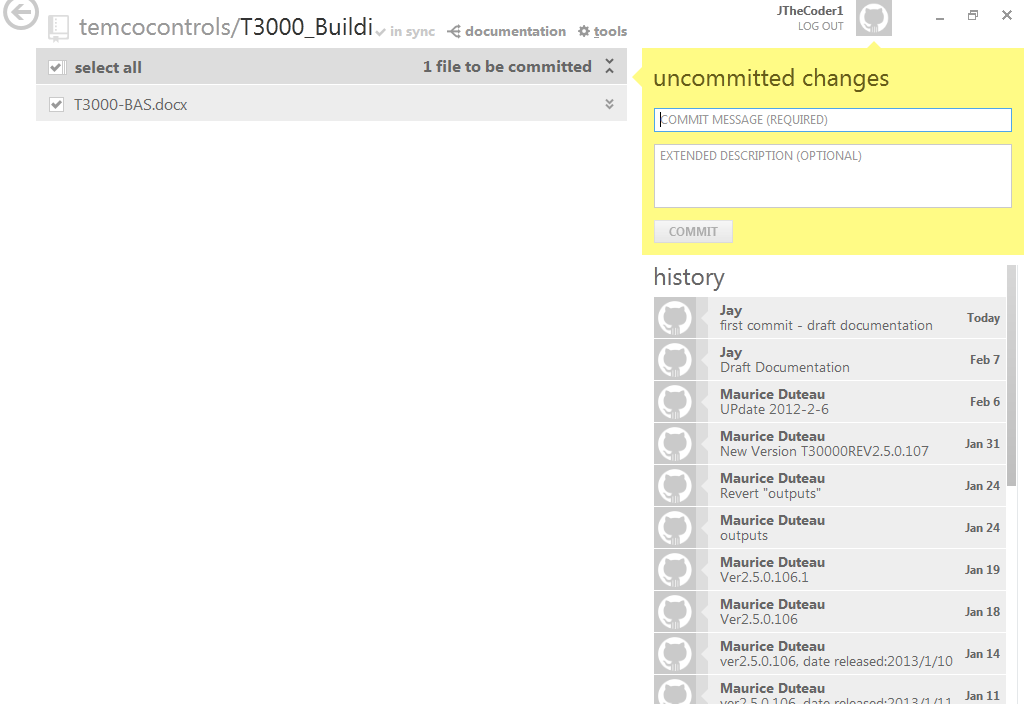
1. If you click on the small blue arrow on the repository, you would be taken to the following screen.



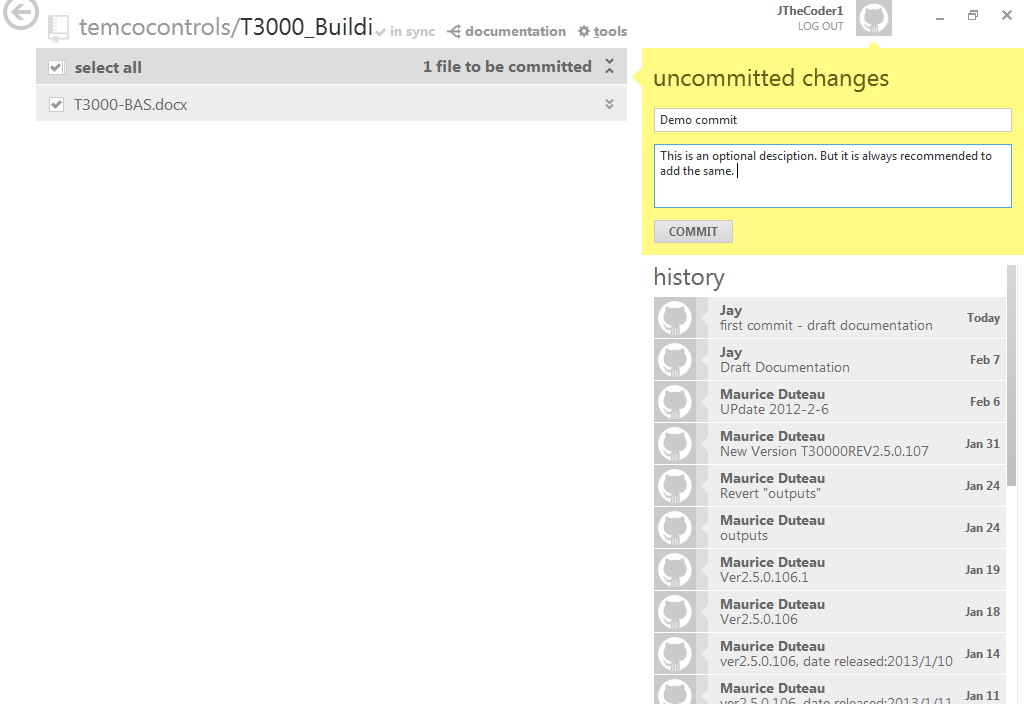
1. If you click on any of the links on history pane on right side, you get all the files that are there in the repository for that version.



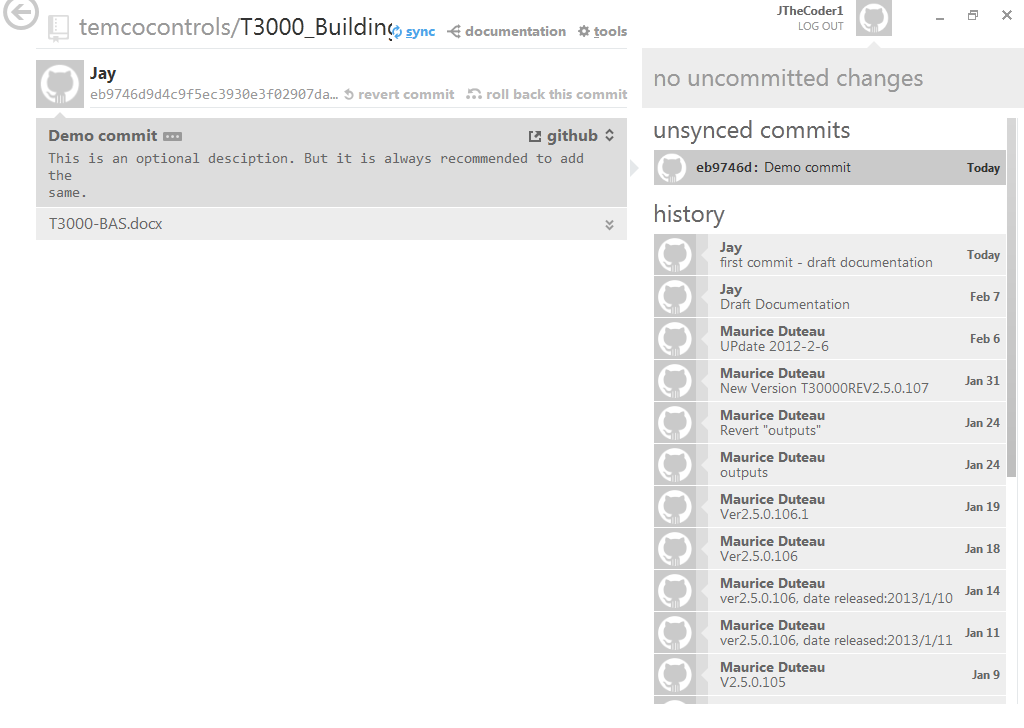
1. Once the file is modified in the repository, it appears on the screen as follows. I have the current file, which I am typing write now as the one which is to be committed to the repository. The same is shown on the screen.



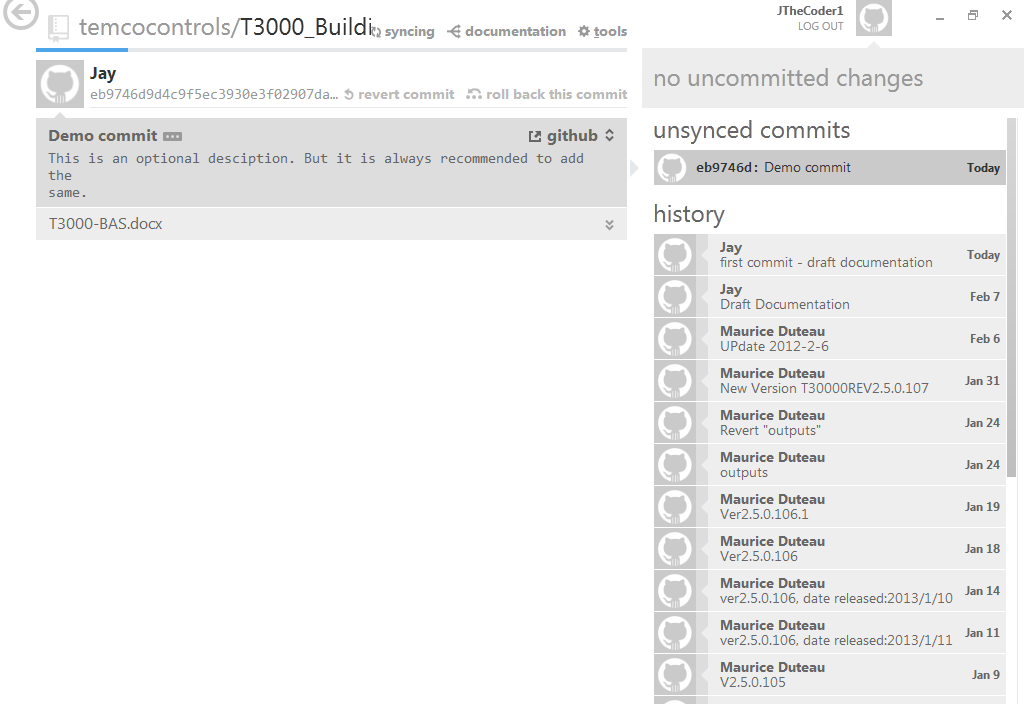
1. You can commit the change in the repository by adding a commit message and optionally an extended description. And then click on the COMMIT button given in the yellow area on right.



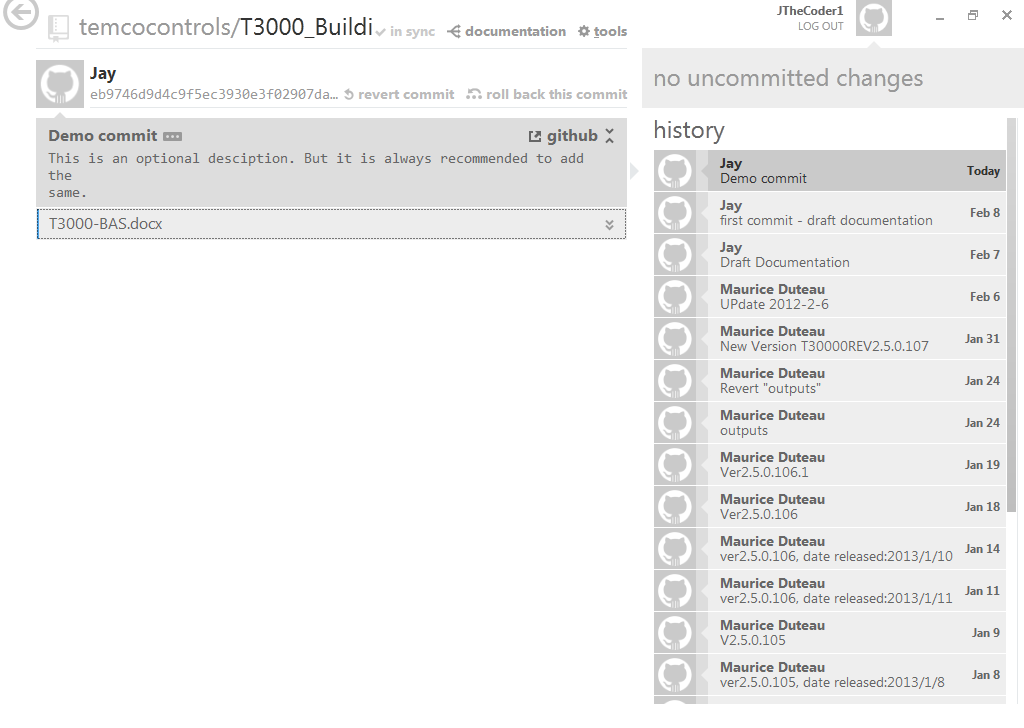
1. Once the COMMIT button is clicked, changes are committed to the repository. But they are not in sync with the repository on the github server. To push the changes to the github server, click on sync button as shown in blue in the following figure.



1. Following figure shows sync in progress.



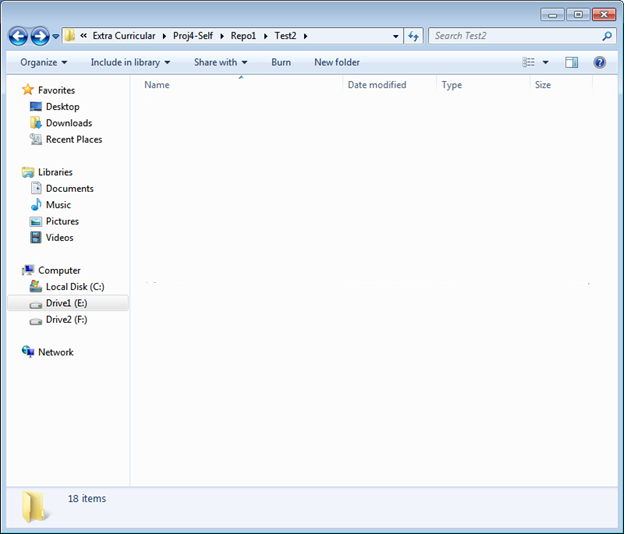
1. Once in sync, the same is shown as follows.



1. You can work with github and repository in this manner.

# Procedure to Merge

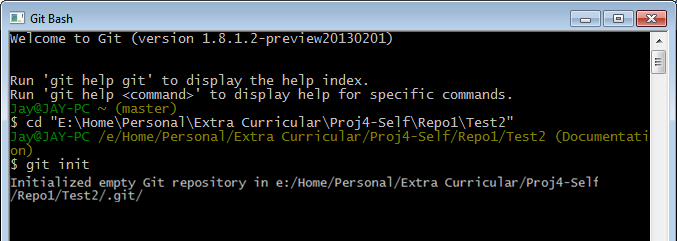
1. First decide a folder on your PC where you want to create local copy of your git repository and git working directory.
2. On my PC, this folder is “E:\Home\Personal\Extra Curricular\Proj4-Self\Repo1\Test2”.



1. Open Git Bash and type following commands.

$ cd <”path of folder”>

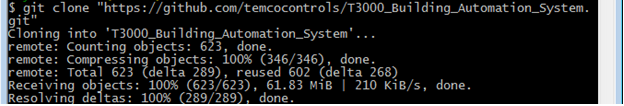
$ git init



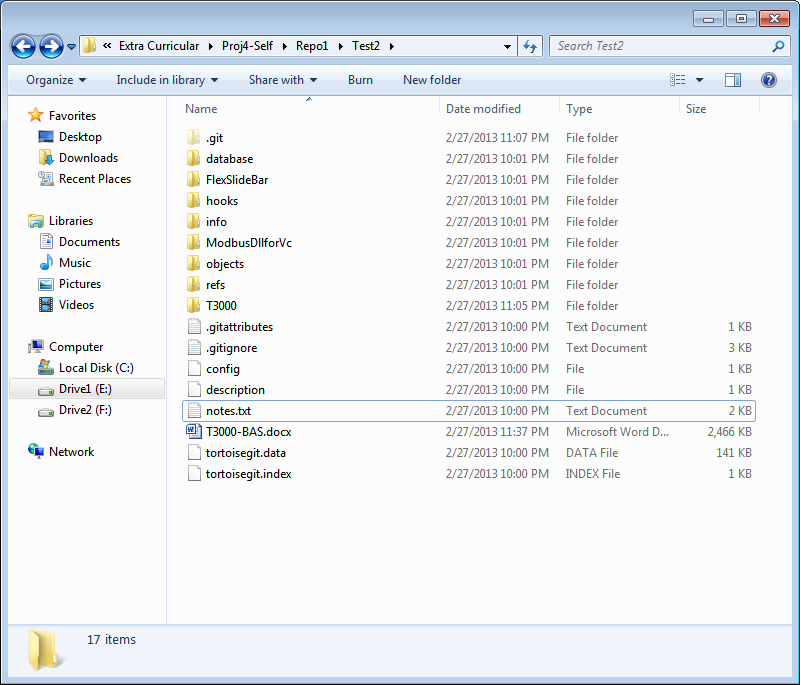
1. Next step is to clone repository of T3000\_Building\_Automation\_System from github server. To achieve this, type following command.

$ git clone “https://github.com/temcocontrols/T3000\_Building\_Automation\_System.git”

Upon executing this command, a message similar to the one shown in screen shot below appears. It might take 5 to 15 minutes depending on internet speed to clone complete cloning of repository.



1. Once the cloning is complete, folder on PC looks like as shown below. There would be a hidden sub-folder named “.git” which is local git repository. Outside of it Is working directory.



1. Now, use following git commands to do tasks as desired. I am documenting normal steps that would be followed by development team (Alex) and documentation team (Jay).

* Both documentation and development teams will have to clone git repository from github.com.
* Run following commands to determine the branches present in the repository. Remember that once cloning is done, all these commands would work on local repository and not on repository on github server.

$ git branch

This command lists all the branches present in the local repository. In the list which appears following this command, there would be one branch with \* marked. This is the current branch of the repository.

For basic concepts of git branch read following material:

<http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is>

For complete tutorial on branching and merging refer to chapter 3 of Git Book.

<http://git-scm.com/book/en/Git-Branching>

Let’s say there are two branches:

Master and Documentation.

* Documentation team shall move to Documentation branch by executing following command.

$ git checkout Documentation

This command loads all files and folders of repository to working directory.

* Documentation team shall add its comments to all the source code files. Once, a particular level of commenting is reached, changes shall be committed to git repository by executing following commands.

$ git add “\*.\*”

$ git commit -m “commit message”

‘git add’ command adds files to the staging area and marks them for the commit (To understand this concept, read git theory from Git Book: <http://git-scm.com/book/en/Getting-Started-Git-Basics>). ‘git commit’ command actually commits staged files to repository.

* Once the Documentation branch is updated, documentation team shall push it to github for the world to see it.

[I will update this procedure soon. I am still having some doubts on this, so trying to clarify from various sources of mine.]

* Development team can fetch the contents from github server using following command.

$ git fetch

This command fetches all the new changes that have been pushed by documentation team to the local repository of development team.

* Then development team may switch between master and documentation branches and go through the changes – that is, the commenting done by documentation team. It may decide to merge the changes of documentation branch to master branch using following command.

$ git checkout master

$ git merge documentation

These commands merge documentation to master branch. Now, the changes done by documentation team should have been reflected in master branch.

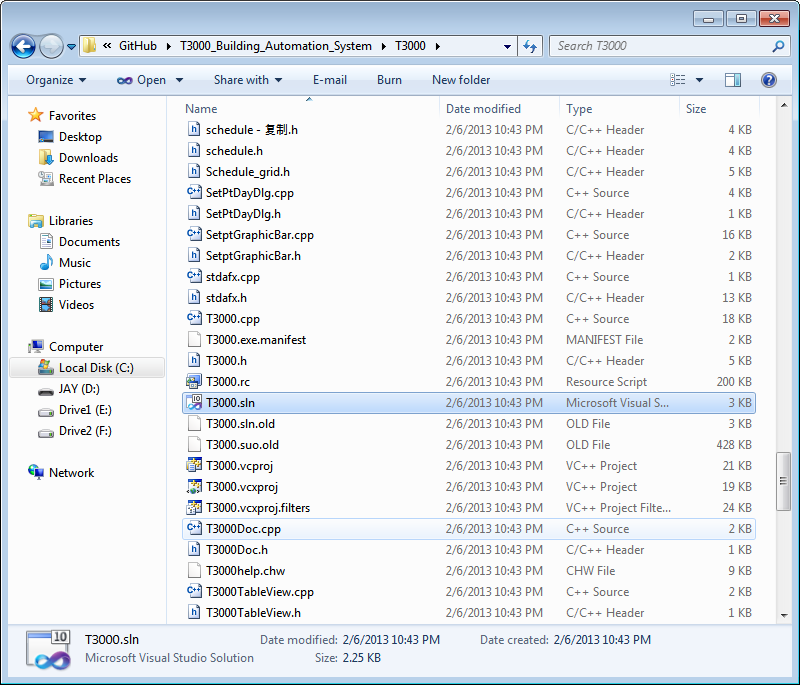
* Once, this is done, development team shall push the changes to github and inform documentation team.
* This way, both teams can collaborate.
* [This procedure may change in near future as roughness would be smoothened out. Till then keeping this statement as it is. ]

# Compilation on Microsoft Visual Studio 2010

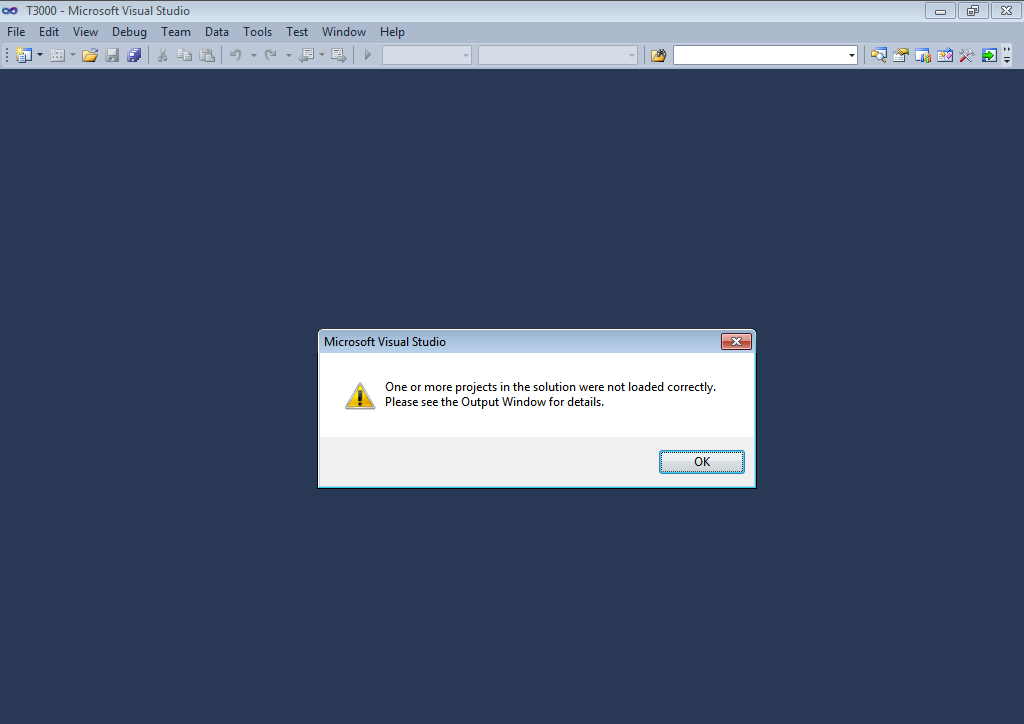
1. Install MS Visual Studio 2010 on your PC.
2. Start MS Visual Studio from the main start menu.



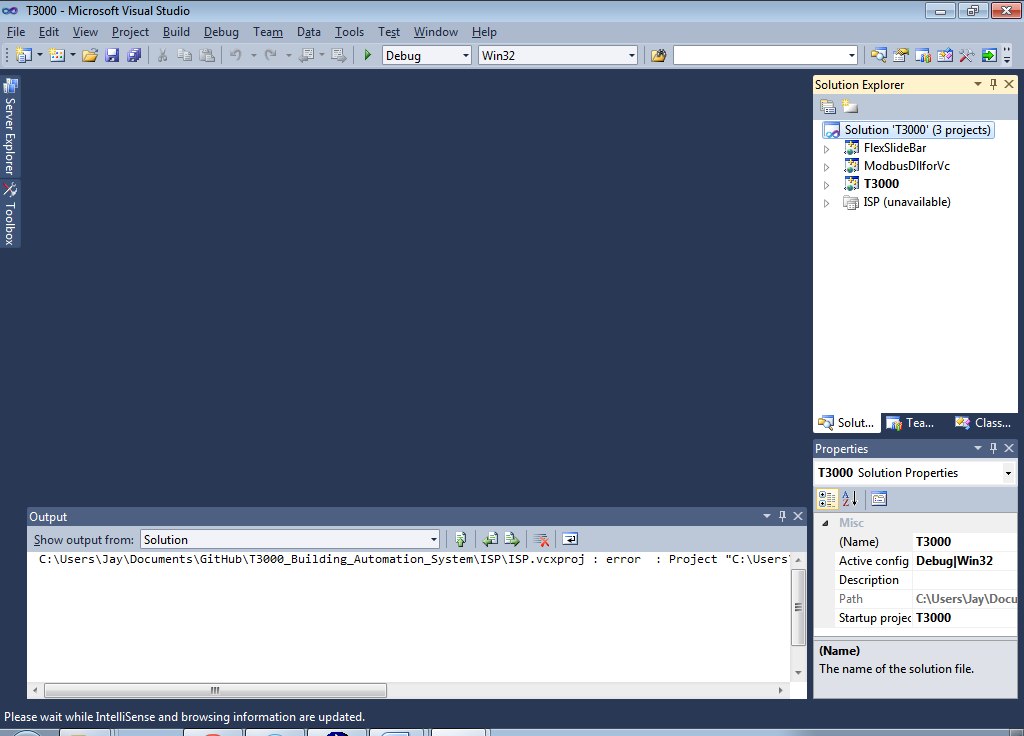
1. Go to the repository directory. Go to T3000 folder. Click on T3000.sln.



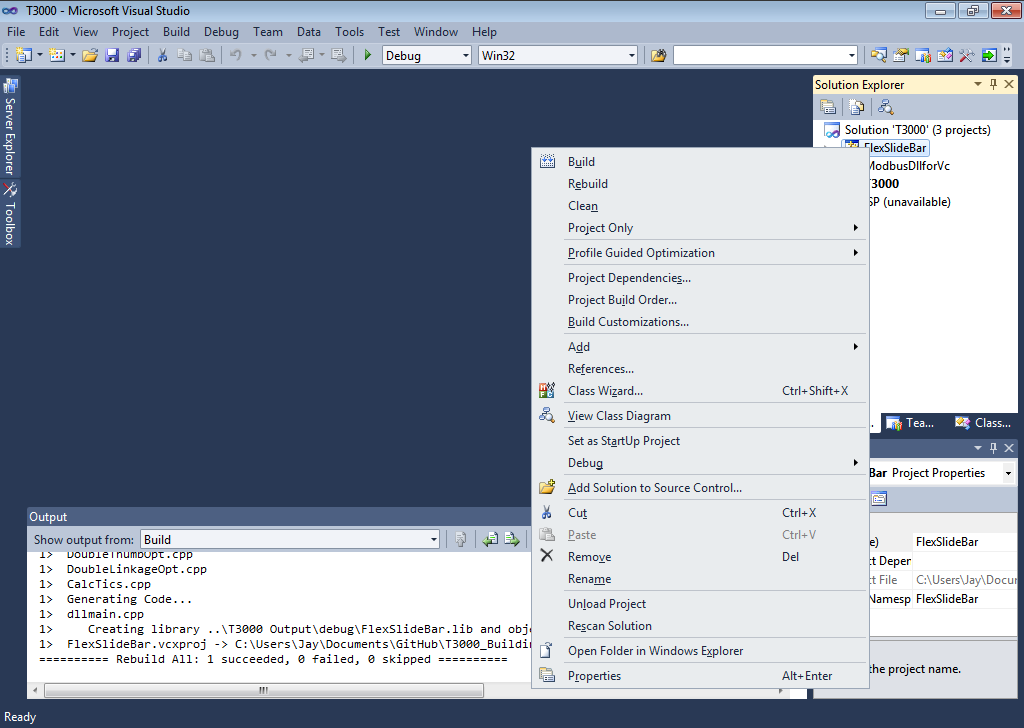
1. An error window appears as follows. [Right now, not sure why is this error coming? Anyway, capturing it in this doc as of now. We will remove this once the issue is solved. ]



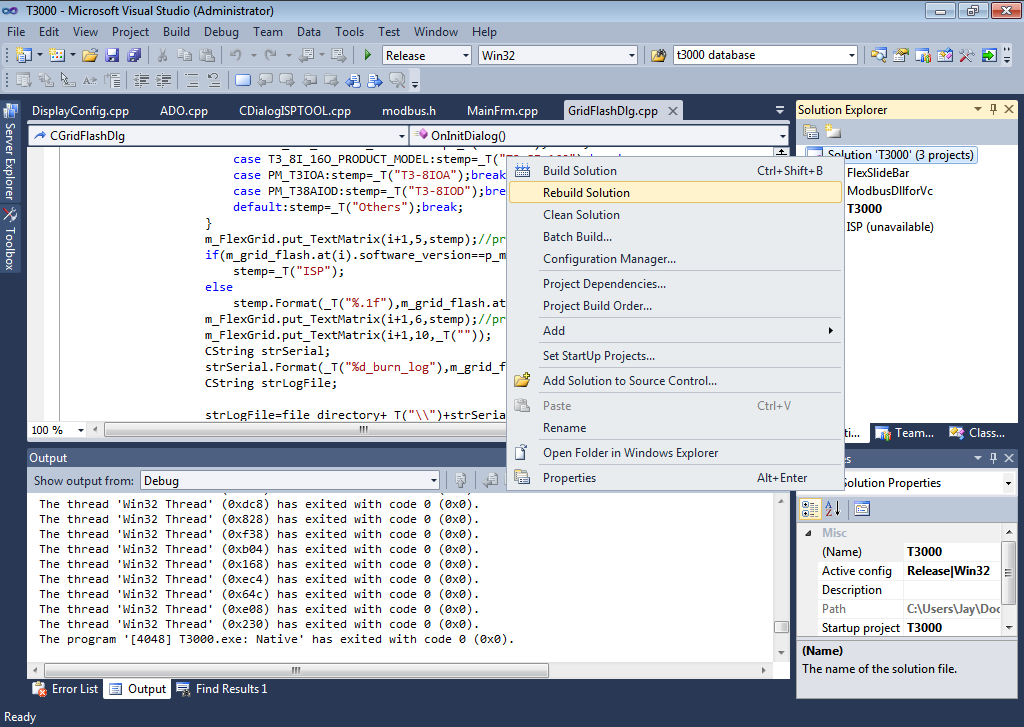
1. The following screen appears.



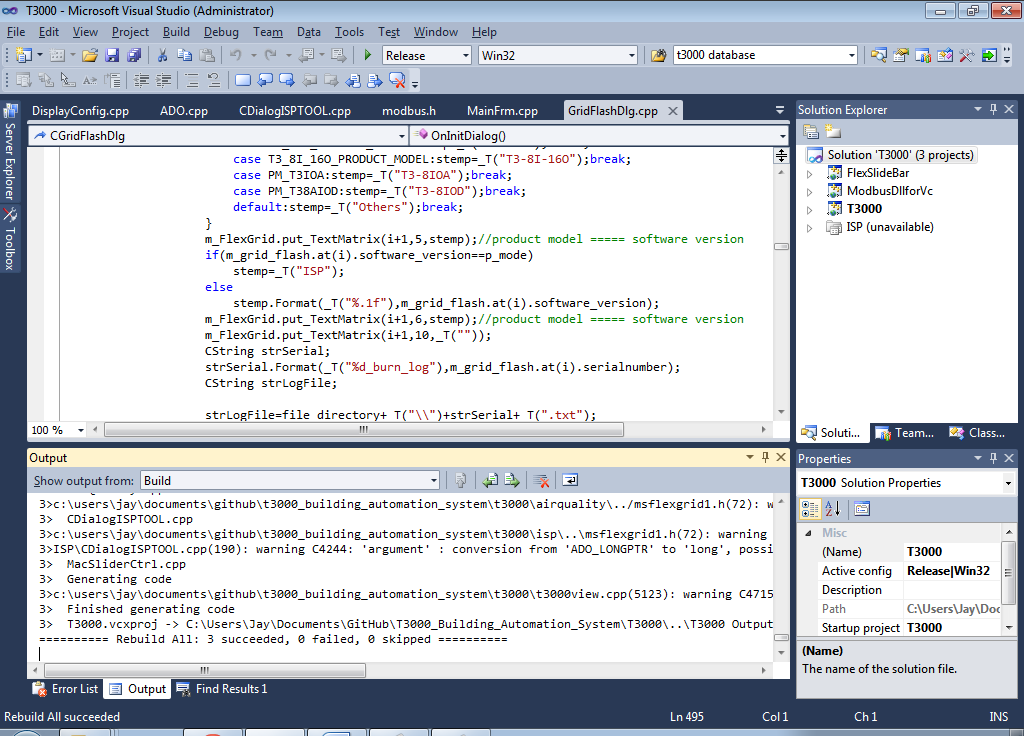
1. Right-click on FlexSlideBar project. And click Rebuild. This shall compile FlexSlideBar project.



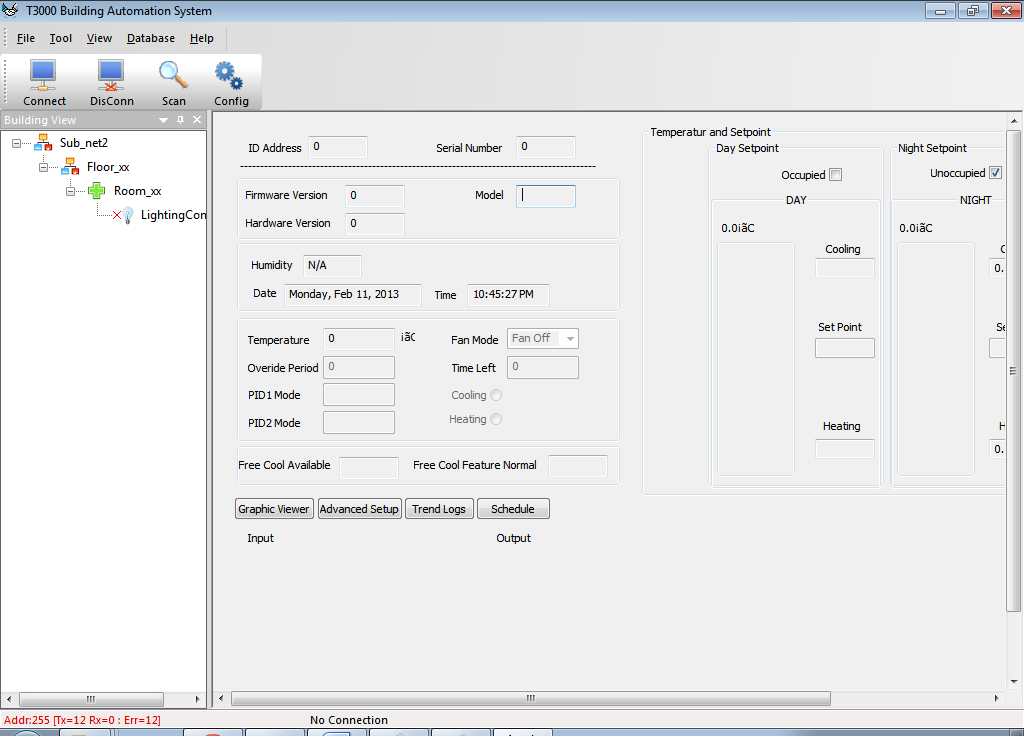
1. Similarly compile ModbusDllforVc project by right-clicking it and clicking Rebuild.
2. Similarly compile T3000 project by right-clicking it and clicking Rebuild.
3. Another way to compile all the projects is to right-click on solution T3000. And click on Rebuild Solution.



1. Once the code is compiled successfully, Output window of IDE shall show some text as shown below.



1. Once the code is compiled and it is Run, following application runs.



# Git commands summary

1. git config:

To set configuration parameters of git.

1. git help

To get help from man pages for git or any of its specific commands.

1. git clone

To clone a git repo from a remote repository.

1. git status

To get status of files that are to be staged / committed to the repository.

1. git add

To mark files to be added to the repository in the next commit.

1. git commit

To commit files to the repository.

1. git diff

To check differences between two versions.

1. git log

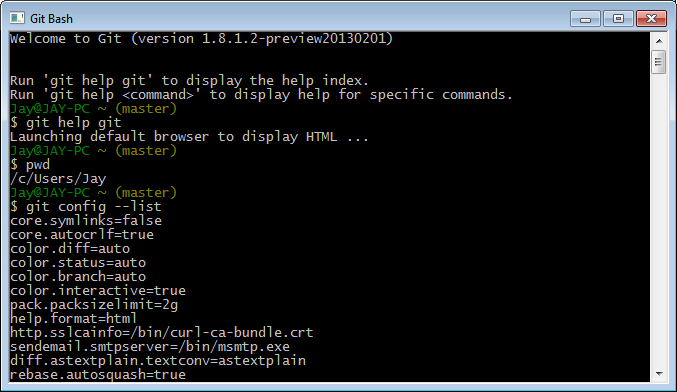
To view commit history.

1. gitk

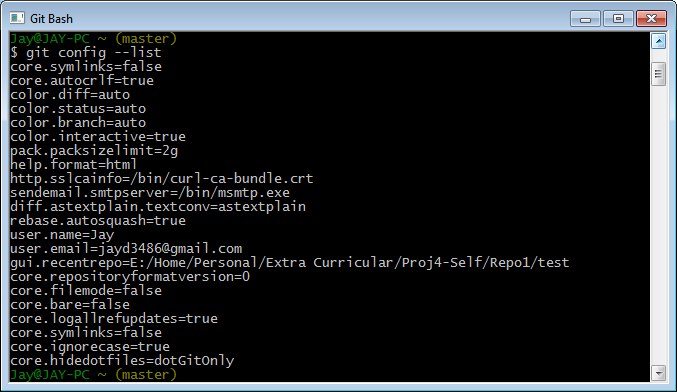
To invoke a GUI to visualize history.

# Cloning a repo from github.com using Git Bash Shell

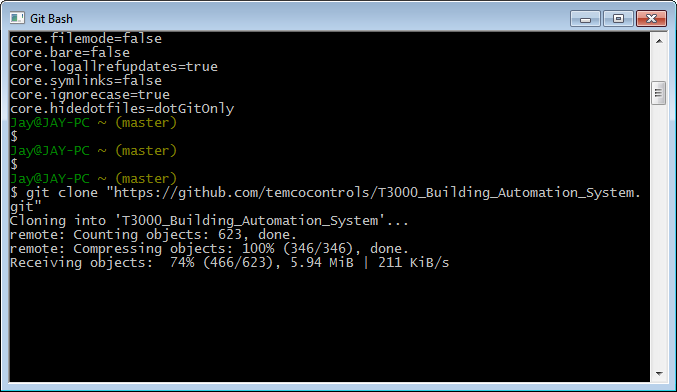
1. Use pwd (present working directory) command to know your working directory.



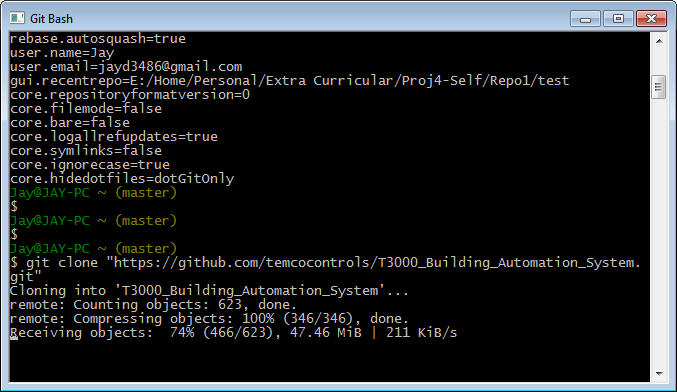
1. You can use git config –list to know present configuration settings of git.



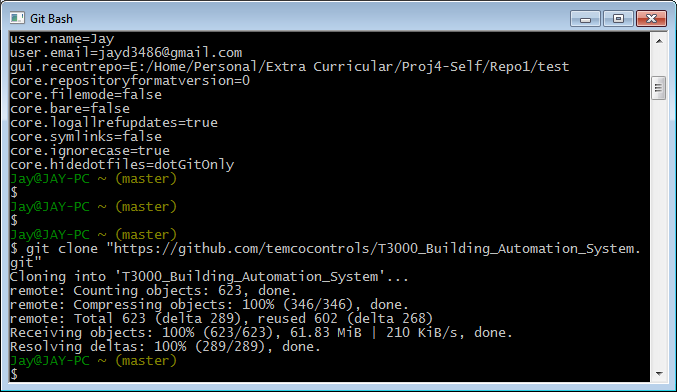
1. Use clone command as shown below to clone T3000 project from github website.



1. Continuation of the same.



1. Cloning done.



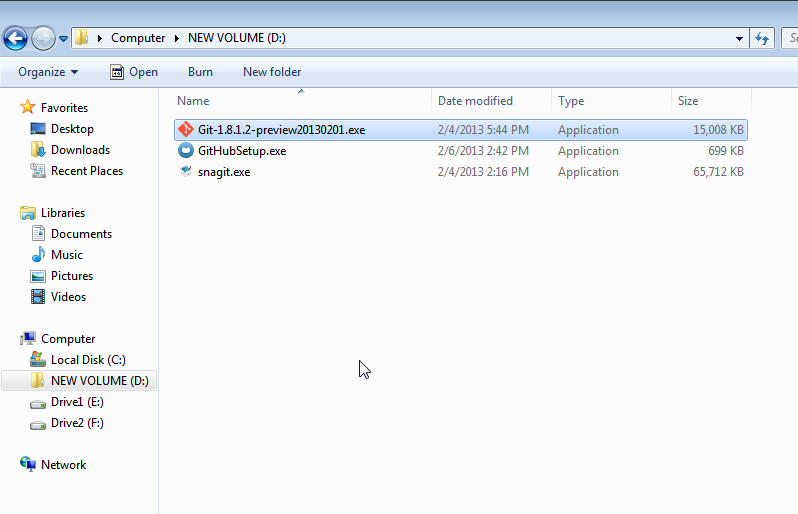
# TortoiseGit

Setting up of TortoiseGit involves following steps:

1. Installation of msysgit, the windows version of Git
2. Installation of TortoiseGit
3. Sign up for a github account
4. Setting up SSH key
5. Working with repository

# Installation of msysgit

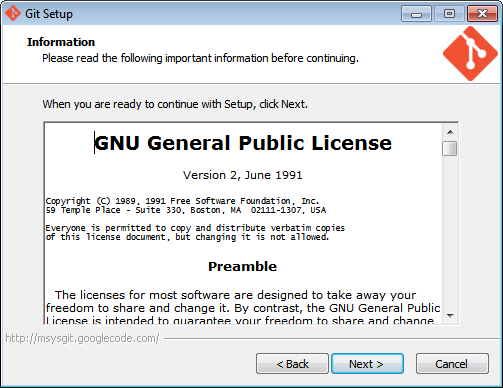
1. Download git installer.
2. Double click on the setup file.



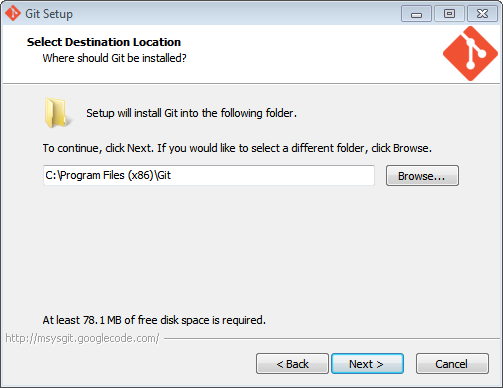
1. Following dialog box opens. Click Next to continue.



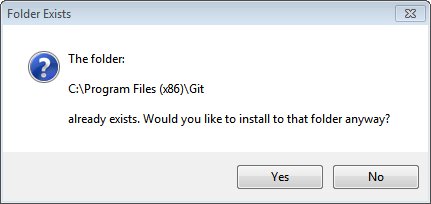
1. Read GNU General Public License. Click Next.



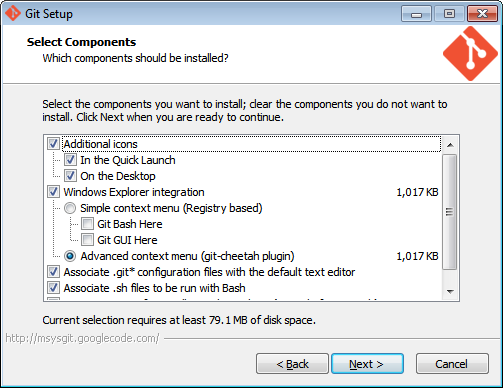
1. Select destination location. This is the directory where you want to install git. Normally, default directory can be kept as it is. If you wish, you can change the directory by clicking on Browse button. Here, we will install git in the default directory. (C:\Program Files (x86)\Git). Click Next.



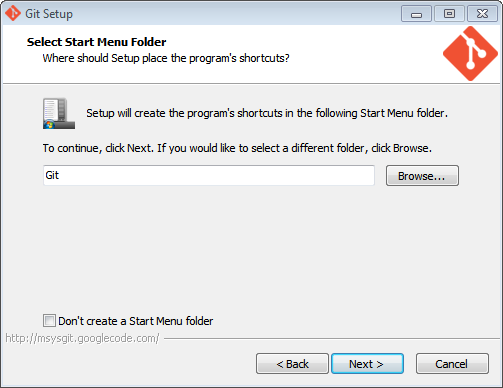
1. If following dialog box appears, click Yes.



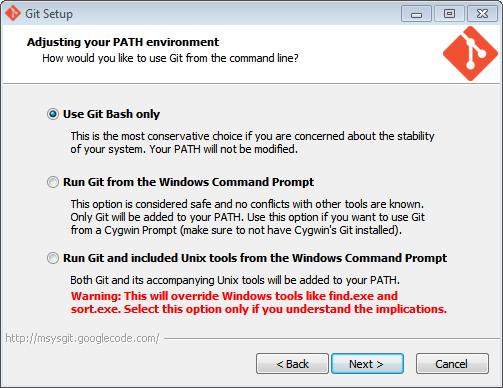
1. Following screen appears. Leave all the default options as is and click Next.



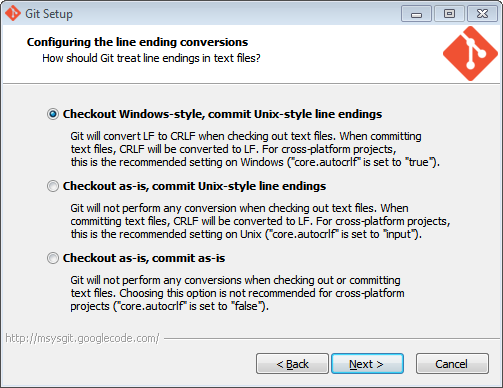
1. Following screen appears. Click Next.



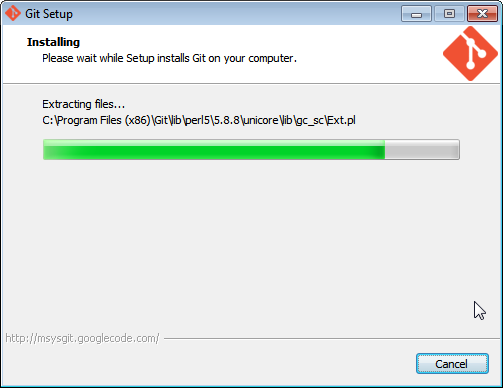
1. Following screen appears. Click Next to continue.



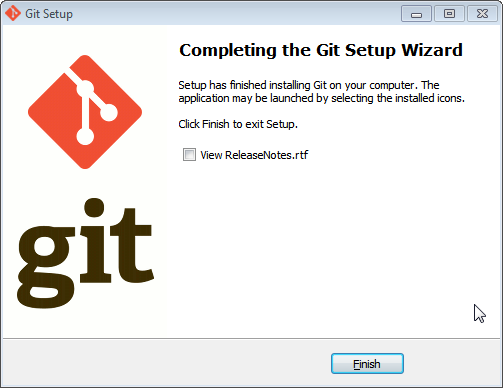
1. Following screen appears. Click Next to continue.



1. Setup will progress.



1. Click Finish to complete the setup.



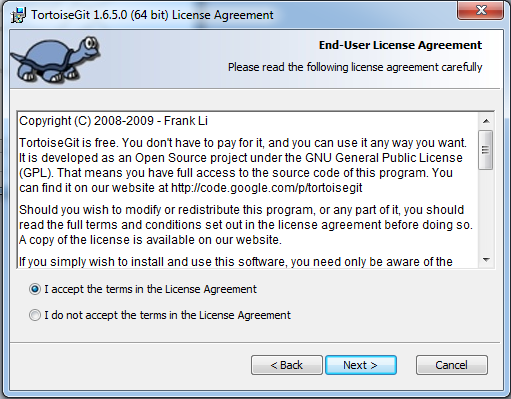
# Installation of TortoiseGit

[TortoiseGit](http://code.google.com/p/tortoisegit/) is a GUI wrapper around git, and provides one of the cleaner user experiences. You'll need to download and install msysgit before installing TortoiseGit.

1. The first screen you'll be presented with is a splash screen.

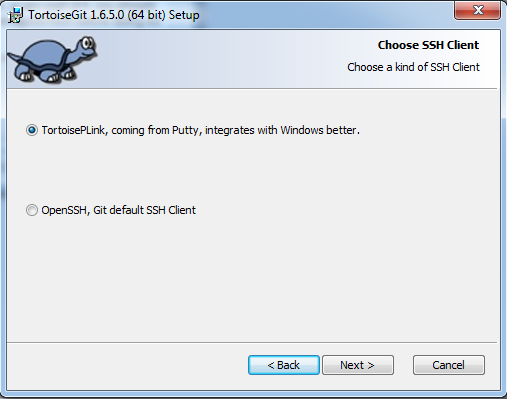


1. Click Next, and EULA will pop up.



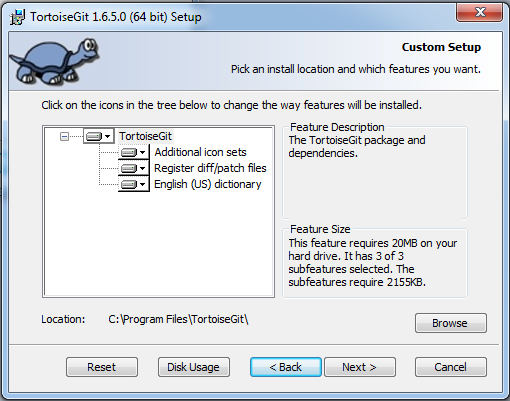
Select ‘I accept…’ and click Next.

1. Following screen appears.



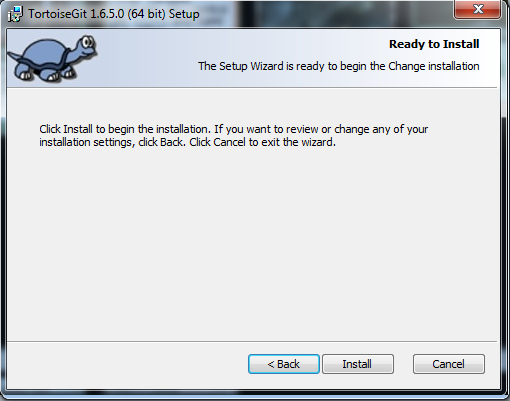
Make sure TortoisePLink is selected, and then press Next.

1. Screen for custom setup appears.

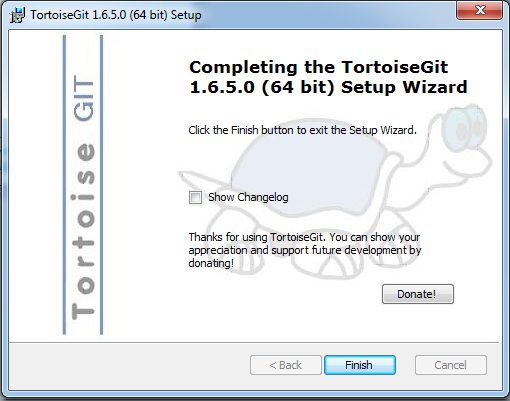


Click Next.

1. Following screen appears. Click Install button.



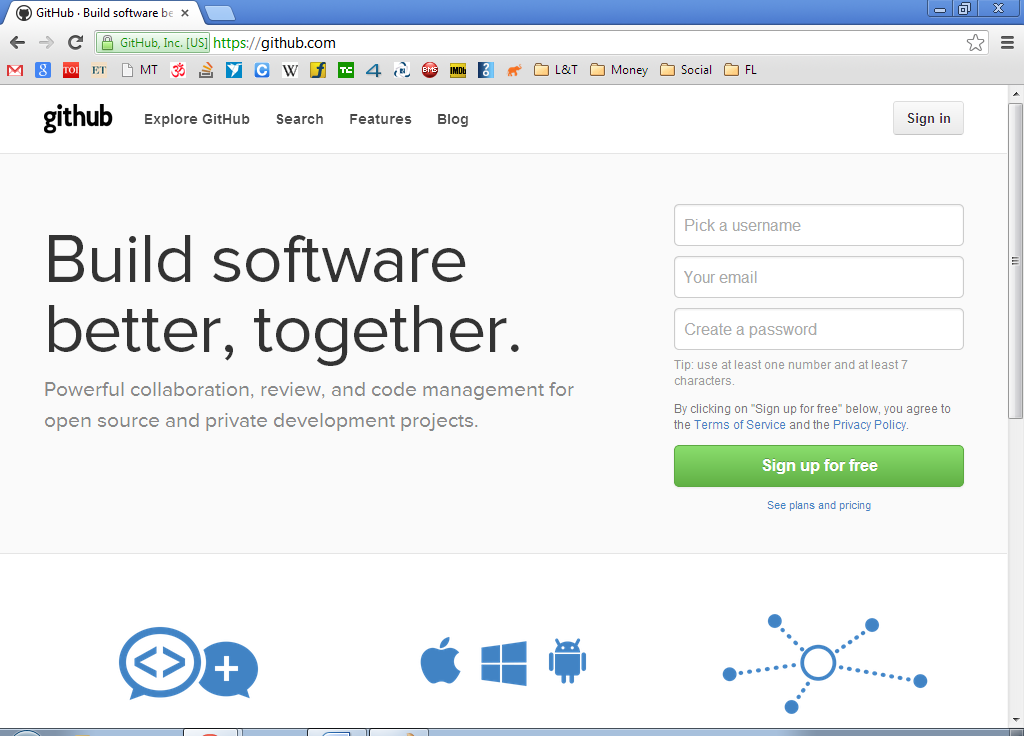
1. TortoiseGit gets installed and following screen appears. Click Finish.



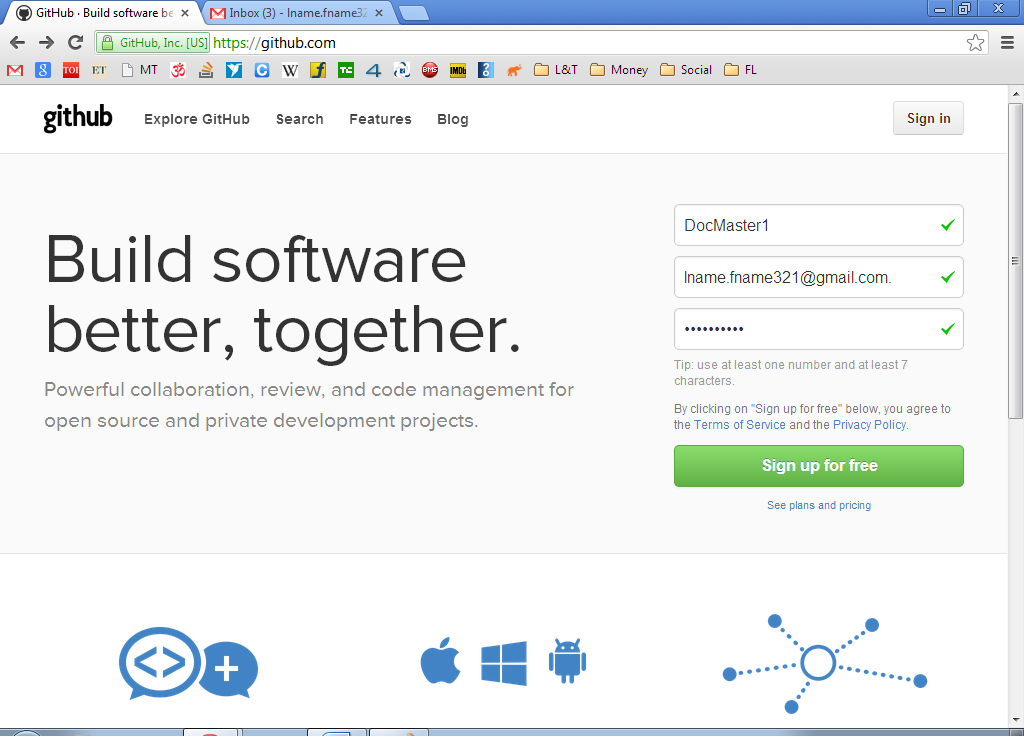
Your machine is now all set-up for playing with code on Github. You just need a Github account.

# Sign up for github account

1. Go to github.com website.



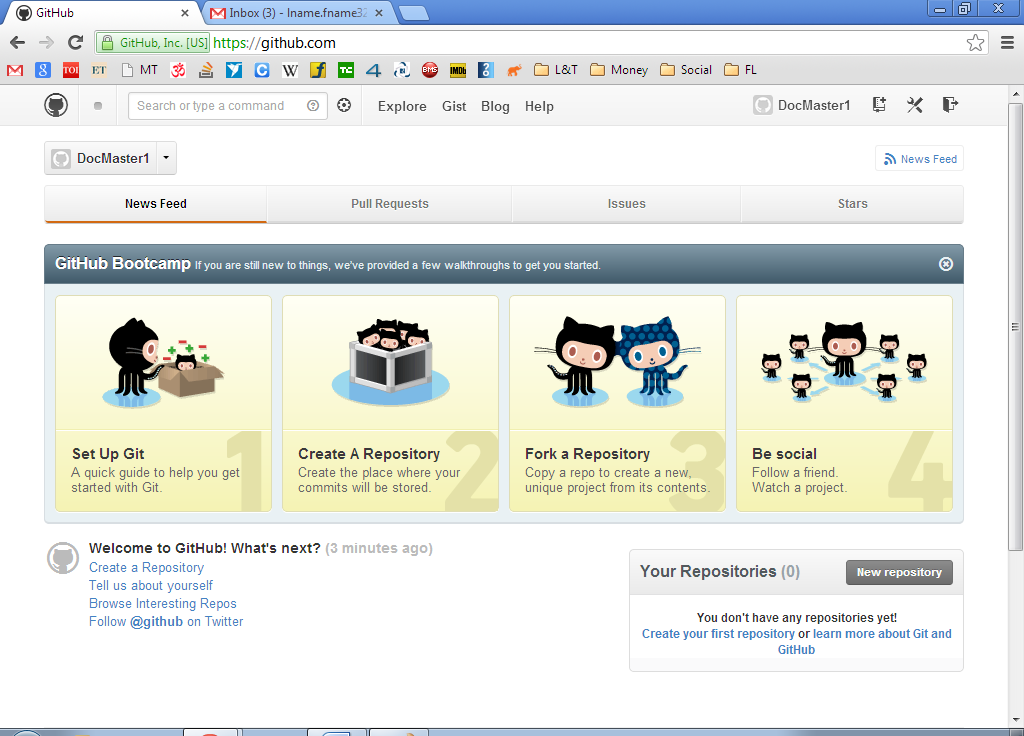
1. To setup github account, enter username, email id and password. Click ‘Sign up for free’.



1. If there are any warnings, github shows them and takes you to the next page. Click on ‘Create an account’ button.



1. Now your github account is ready and you are good to go.



# Setting up SSH key

To use your Github account on a Windows machine, you'll need to generate a new [RSA key](http://en.wikipedia.org/wiki/RSA_(algorithm)). You'll also need the following tools installed:

* msysgit
* TortoiseGit

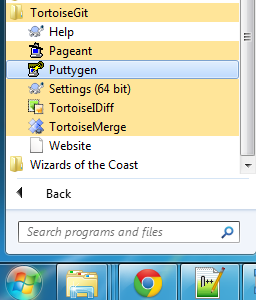
1. Generate a new RSA key

RSA is an [asymmetric encryption](http://en.wikipedia.org/wiki/Asymmetric_encryption) algorithm used to authenticate users without using passwords. It's used instead of passwords simply because it provides a mechanism for verifying who you are without giving the other end of the connection information they can use to impersonate you.

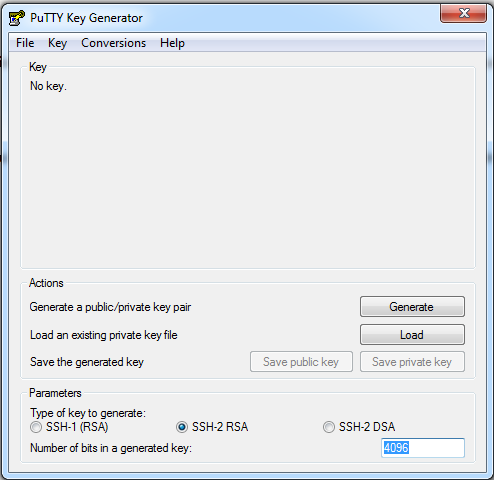
An RSA key consists of two halves: a public key and private key. You give your public key to other sites, which allows them to verify (using a lot of math) that you are the owner of your private key. You must keep your private key private.

Since a private key can theoretically be stolen, it's encrypted with a password (in RSA terms, a "passphrase"). So you need to make sure you both keep the private key in a safe place and remember the passphrase it's encrypted with.

To generate a new RSA key, you'll need to use a tool called Puttygen. It's installed along with TortoiseGit, and you may be able to find it in your start menu:



1. Once it is running, you will be presented with following screen:



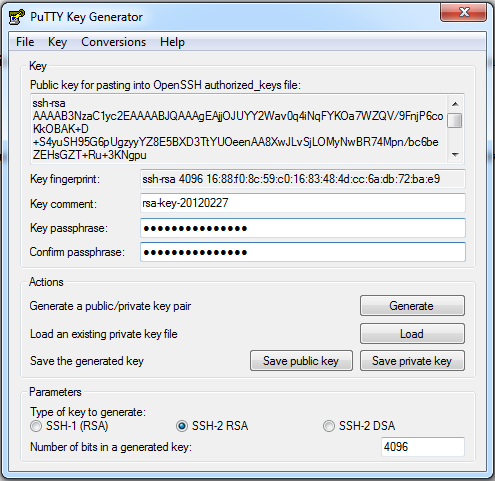
Change the number in the lower-right hand corner, labeled with Number of bits in a generated key, to 4096. This is the current acceptable length for RSA keys. Once this is done, hit the Generate button.

1. Following screen appears. As indicated in the screen, you are supposed to move mouse over blank area of the screen till progress bar reaches its 100 percent value.



The Puttygen program does not use Window’s built-in cryptographically secure random number source. So, this is the fallback option to obtain entropy.

1. Following screen appears.



Do not close this window until the end.

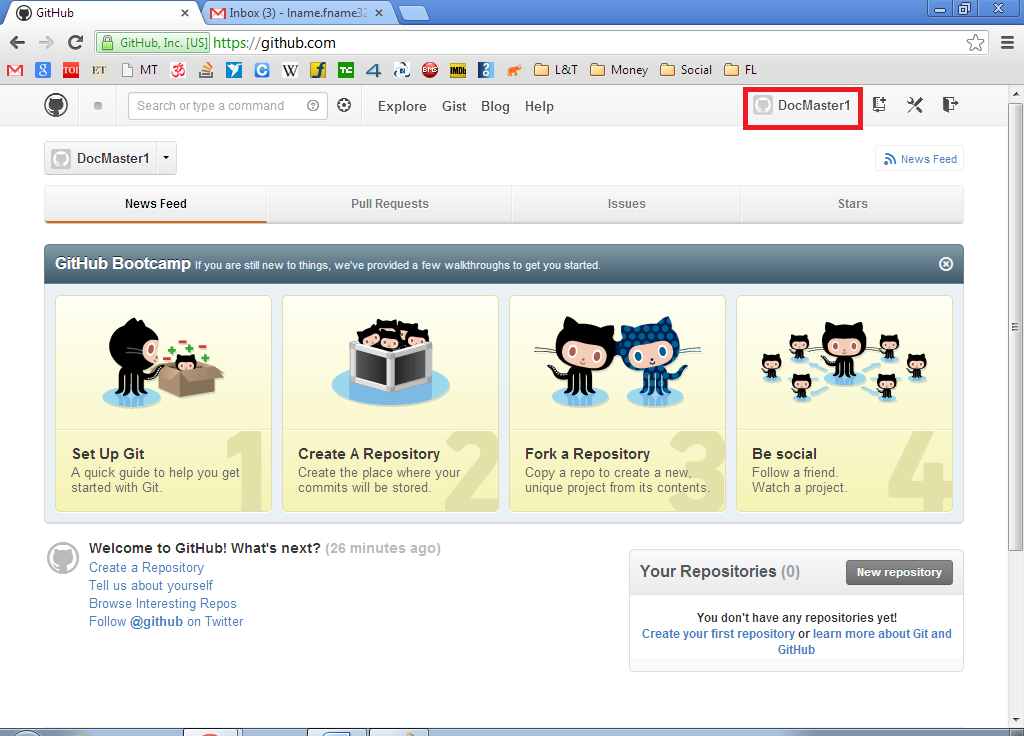
Enter your chosen passphrase into both the ‘Key passphrase’ and ‘Confirm passphrase’ box.

Then, click Save private key and save it somewhere safe.

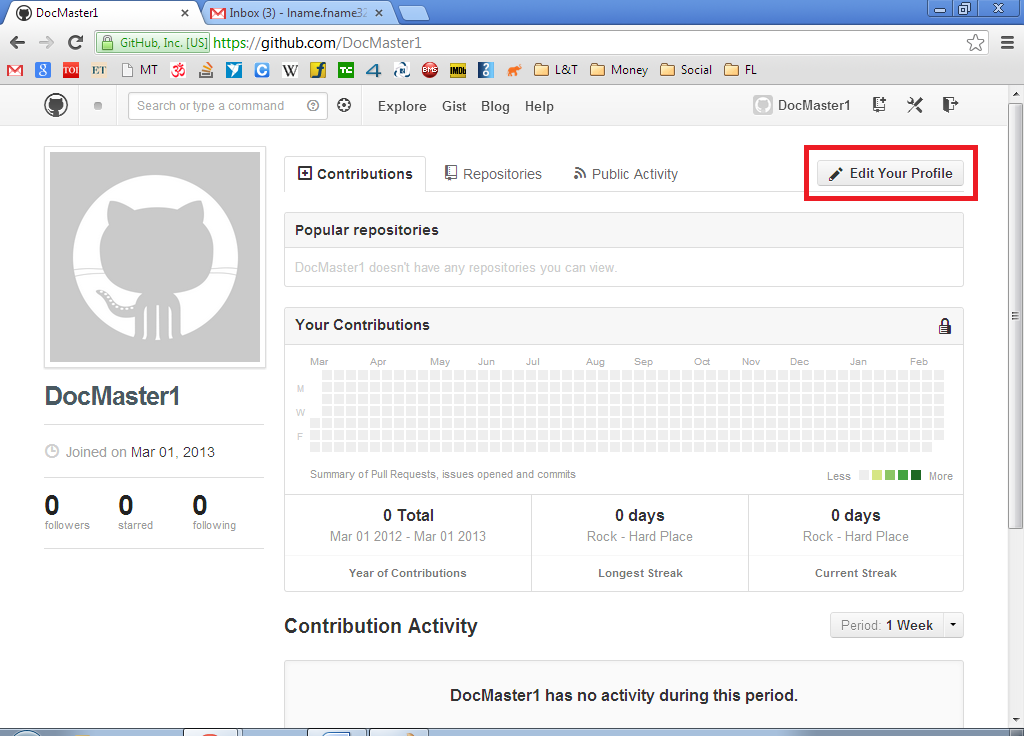
Don't close the window yet. You'll need all of the text in the Public key for pasting... box for the next step.

1. Give your public key to github

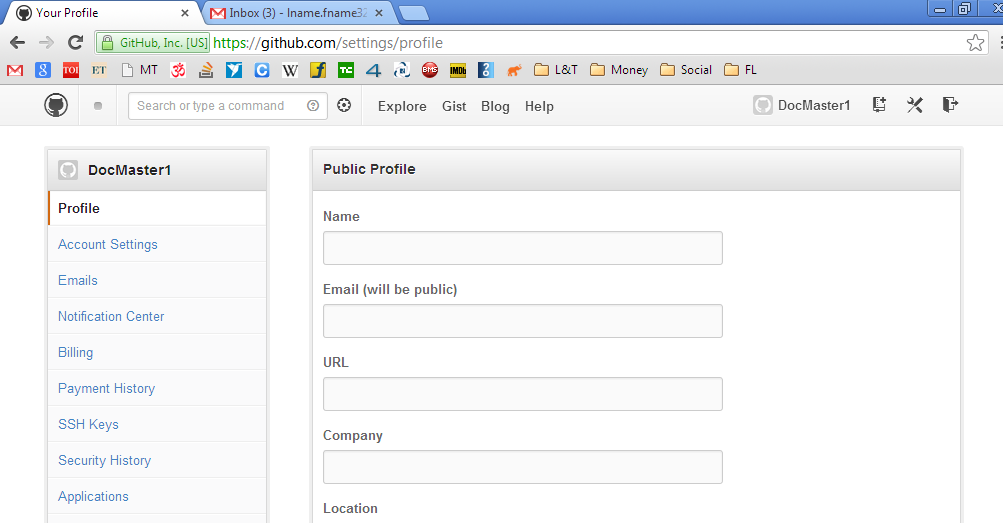
Open web browser. Log into your github account. In the top right corner click your username.



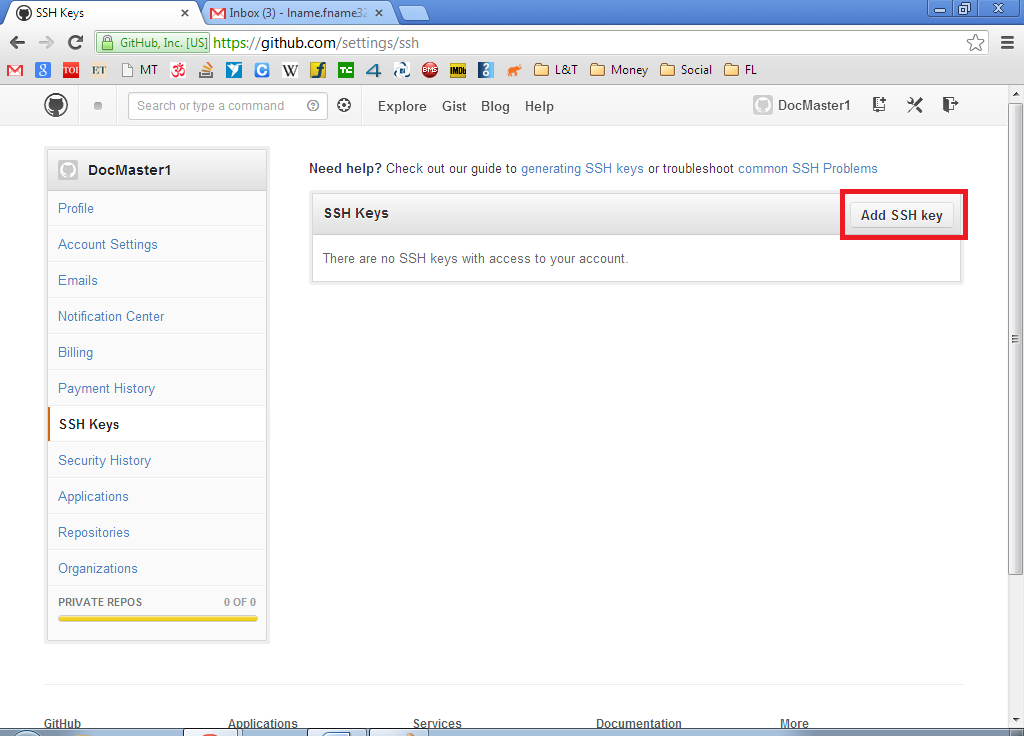
1. From the screen that appears, click on ‘Edit your profile’.



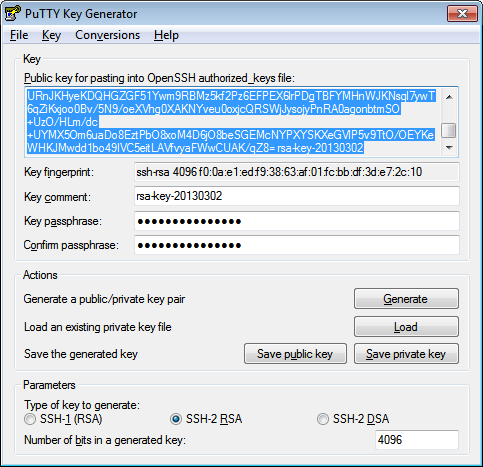
1. Then from the menu on left, click ‘SSH Keys’.



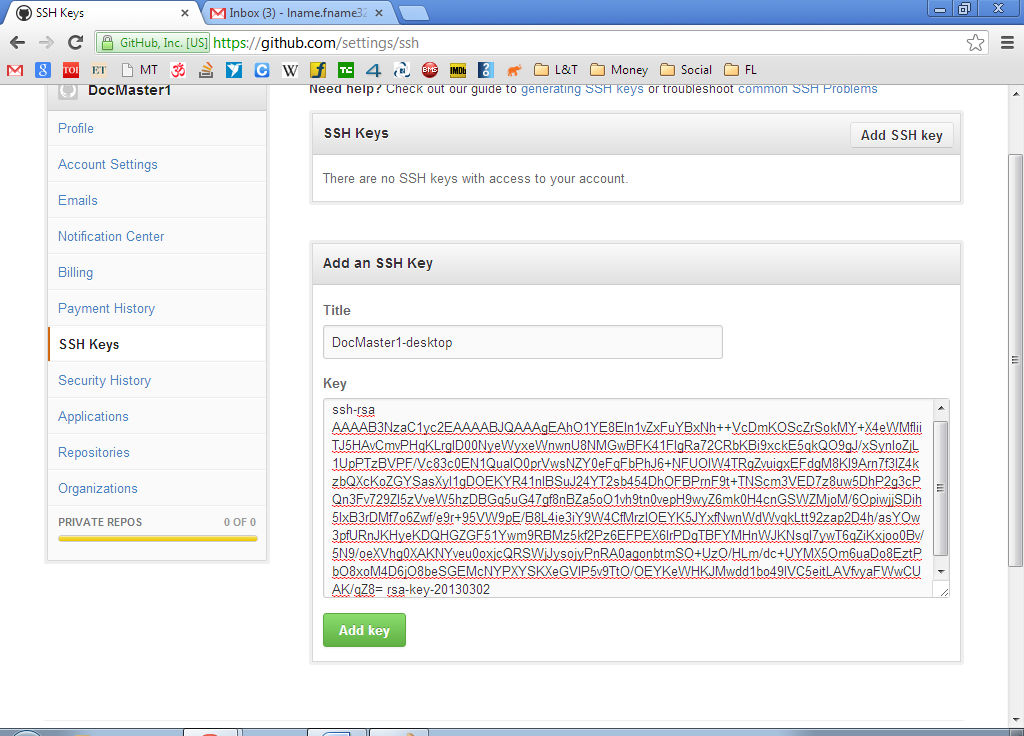
1. Click on the button ‘Add SSH Key’.



1. Go back to Puttygen window and copy text as shown below. Right click -> Select All would be a good option.



1. Paste copied text in the textbox labeled Key. Also, enter a Title.



Finally click ‘Add Key’.

That’s it. Make sure to keep your private key somewhere safe. And do not forget your passphrase.

# Github Workflow

Before starting to learn the workflow, make sure you have following things:

1. Installed msysgit, the windows version of Git (section [6.1](#_Installation_of_msysgit))
2. Installed TortoiseGit, a friendly GUI wrapper around msysgit (section [6.2](#_Installation_of_TortoiseGit))
3. A github account setup with public half of your private key (section [6.3](#_Sign_up_for) and [6.4](#_Setting_up_SSH))
4. Private key and passphrase (section [6.4](#_Setting_up_SSH))

**A word about git and github:**

Git is a decentralized [version control system](http://en.wikipedia.org/wiki/Revision_control). The "decentralized" bit sets it apart from other version control systems like [CVS](http://en.wikipedia.org/wiki/CVS_(software)), [SVN](http://en.wikipedia.org/wiki/Subversion_(software))or [Perforce](http://en.wikipedia.org/wiki/Perforce). "Decentralized" means that there is no central repository -- each developer has a repository (or a few copies) that contains the entire history of everything. This has some advantages:

* You can make commits without being network-connected
* You can commit without worrying about build-breakage
* In a disaster scenario, you're fine unless *every* copy of the repo is wiped out

Unfortunately, these come at the cost of additional complexity.

To simplify things a bit, in most real-world scenarios, one repository is designated as blessed [In our case, this blessed repo would be: **temcocontrols / T3000\_Building\_Automation\_System**]. This blessed repository serves as a central, true copy. All the other repositories (or, "clones") still contain the same version history; the blessed repository simply serves as synchronization primitive to make the workflow cleaner.

You can do a variety of things with a repository. These include --

* Add new commits to a local repository (on your machine)
* Send commits to another repository (on your machine or the network)
* Get commits from another repository (on your machine or the network)

In the setup we're using, each developer interacts with three repositories:

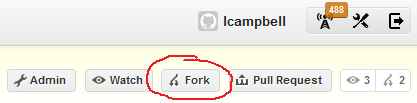
* The repository on your local machine
* Your "fork" on Github
* The "one true repo", which is the central, blessed repository

You're free to do whatever you want with the repository on your machine and your fork on Github. Typically, you'll make commits on your local repository and send them to your Github fork immediately (or, as soon as you're network-connected). Commits going into the blessed repository must go through a review process (using Github's wonderful interface) before they're accepted and merged into the codebase. This process is performed using a pull request on Github, which is detailed further down.

**Forking:**

Right now, we've only got the blessed repository. We need to create both your fork and your local copy set up. First, log into Github, navigate to [our Github page](https://github.com/Uncodin) and select a repository you'd like to work on. I'm going to assume for this case you want to work on **T3000\_Building\_Automation\_System**.

Click the Fork button in the right-hand corner:



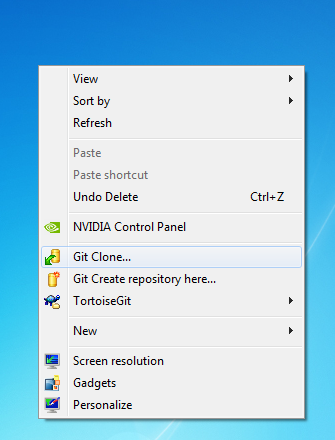
After some hardcore forking action, you'll be greeted with your Github fork. This is your personal copy of the blessed repository.

Assuming your Github username is **Jay** and the name of the upstream repository is **T3000\_Building\_Automation\_System**, your fork will be located at **https://github.com/Jay/ T3000\_Building\_Automation\_System**.

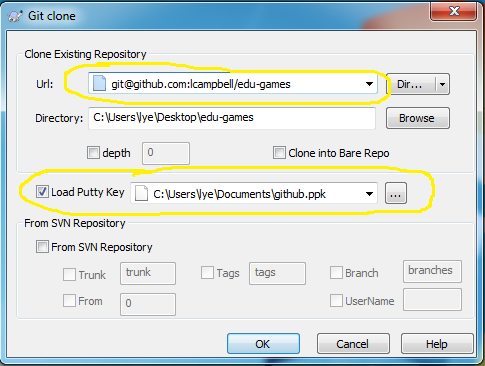
[Remember, these names are just for example. In reality, username Jay has already been taken up by someone else. Here, I am mentioning the same because it would be easier to understand. Also, the usernames and repo names shown in diagrams are different ones. Do not get confused by that. Just have a look at diagram for position of the buttons and conceptual understanding, etc.]

**Cloning your fork into a local repository**

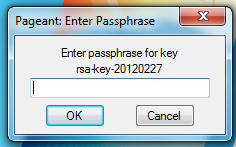
Git doesn't allow you to commit directly to a repository on the network -- you need to make a local copy before you can start doing work. Making a copy, in Git terminology, is called "cloning", and we're going to do it with TortoiseGit. Start by right-clicking on the Desktop (or other appropriate place) to bring up a context menu.



From this menu, select Git Clone... and you'll be greeted with the following dialog:

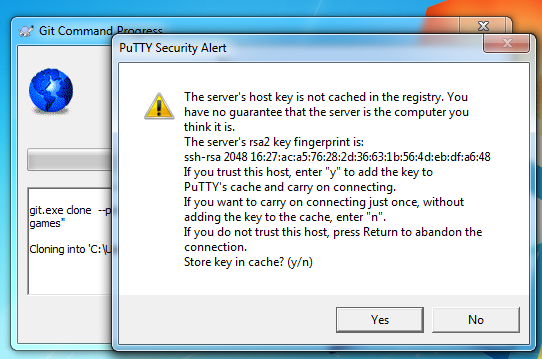


Here, you'll need to enter the URL for your Github fork, and the location of your private key. **Do not enter the URL of the blessed, upstream repository**. Once you've got the appropriate information inputted, click OK, and it will prompt you for your key's passphrase.



Toss in your passphrase and hit OK. TortoiseGit will save your passphrase for an indeterminate amount of time.

If this is your first time connecting to Github's servers, you'll get the following message:

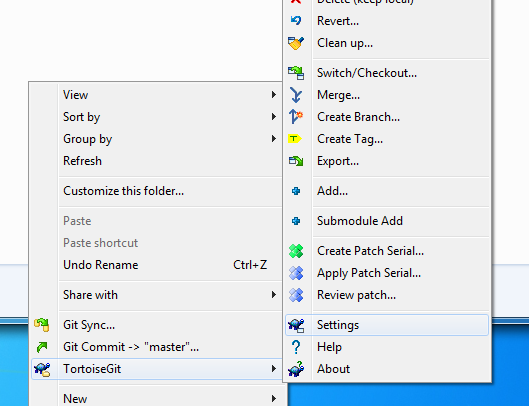


Check to make sure that the rsa2 key fingerprint the dialog tells you is the same as in the screenshot; if it's not, then something is terribly wrong (it's not Github you're communicating with). The reason you get this dialog is due to the nature of public key cryptography. You don't yet have Github's public key. Once you connect once, their public key is saved on your machine and you'll never be prompted again. If their public key changes for some reason, you'll know immediately and Git will cease communication before any damage is done.

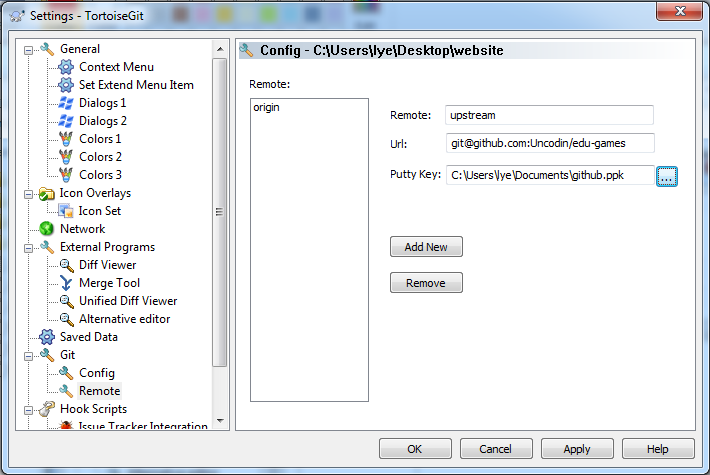
If the key fingerprint is correct, click Yes.

A window should pop up with a progress bar as the repository is copied to your machine (forked repo). Since it contains the entire history it can take a bit of time on a slow network connection.

Once it's finished downloading, we need to take one last step -- we need to tell your local repository about the blessed upstream repository (**temcocontrols/T3000\_Building\_Automation\_System**). To start, open up the TortoiseGit settings by right-clicking in your repository and selecting Settings



Select Remote from the left menu. You'll get this dialog, albiet without the details filled in:



In the fields, put in

* Remote: **upstream**
* Url: The URL to the upstream repository:

**git@github.com:temcocontrols/ T3000\_Building\_Automation\_System**

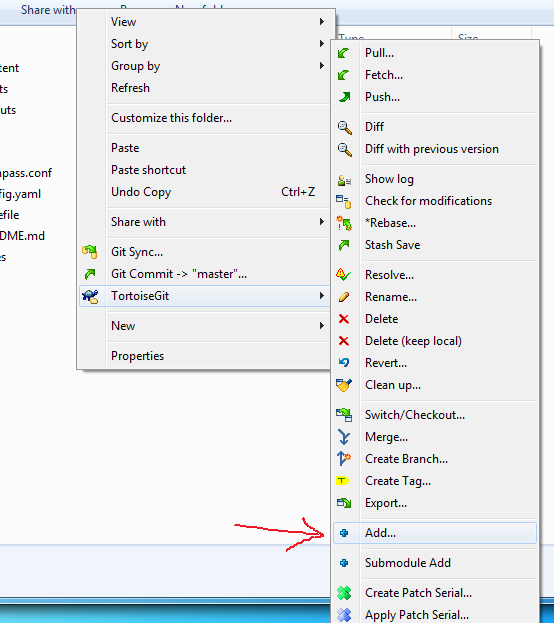
* Putty Key: Your private key

Hit Add New, then OK. You're all set!

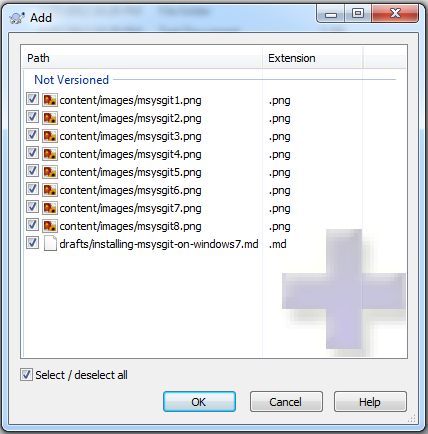
**Making changes, adding commits**

With your fresh clone, look around, make some changes. Add a file or something. Or get some actual work done.

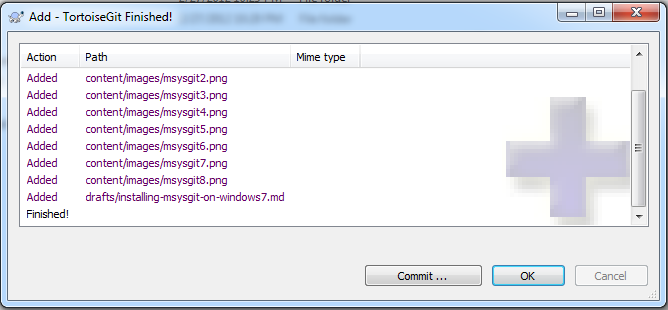
When you've changed something, go to the main folder of your repository and open up the right-click context menu. From that menu, select Add...



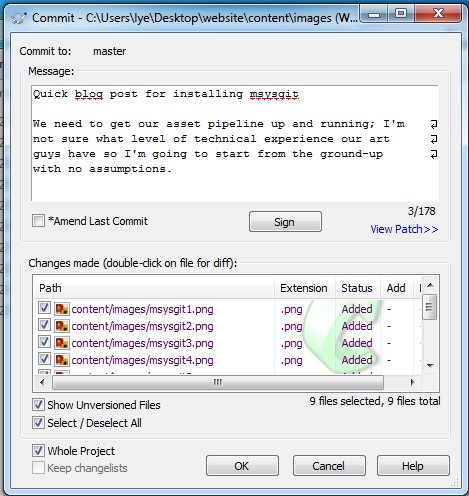
You'll get the dialog telling you which files have been added, removed or changed. Make sure they're all checked and hit OK. Do note that **adding files does not commit them**. You need to do that in a separate step.



Once you've clicked OK, you'll get this:



To hit the previous point home, this dialog gives you a helpful Commit... button to commit the files you've added/changed/removed. If you don't hit commit, you can select Commit from the right-click context menu at a later point in time. For now though, hit Commit... to arrive at this dialog:

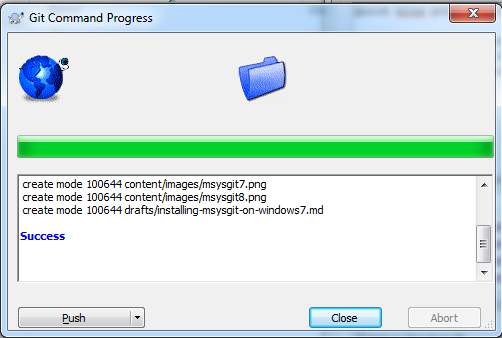


At this point, write a quick blurb about the changes you've made. The first line of your message should be no longer than 80 characters (basically, fit it in the one line of that text box). Each paragraph should be separated by exactly one blank line.

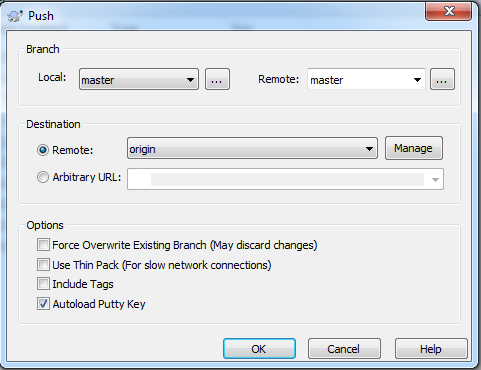
The defaults are usually what you want.

Hit OK to finalize your commit.

You should now be at this dialog:



At this point, we've made a commit on our local repository. We want to send that commit up to our fork [**Jay/ T3000\_Building\_Automation\_System**]; this action is called "Pushing". TortoiseGit gives us a helpful button to push our new commit to our fork. Click Push.

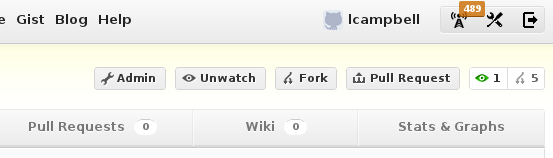


As usual, the defaults are probably what you want here. Make sure that Remote is set to origin, not upstream, then click OK.

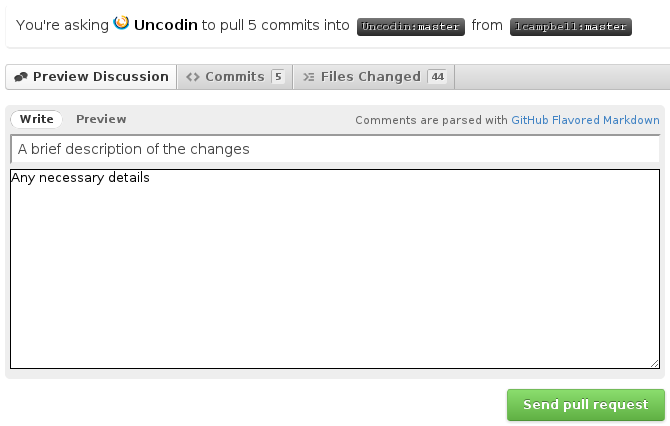
Once your commit is pushed to your Github fork, you can peek at your Github page with your browser to see the changes you've made. Note that they haven't been submitted for review for inclusion into the upstream repository. They're just on your fork.

**Getting changes into the upstream repository**

You've made changes to your Github fork [**Jay/T3000\_Building\_Automation\_System]**, but you need to get them into the upstream repository [**temcocontrols/T3000\_Building\_Automation\_System]** for everyone else to have access to them. This is done by sending a pull request on the Github web interface. Log into your account and navigate to your fork's page. In the upper right-hand corner, there's a Pull Request button.



Click the Pull Request button and you'll be taken to the pull request page.



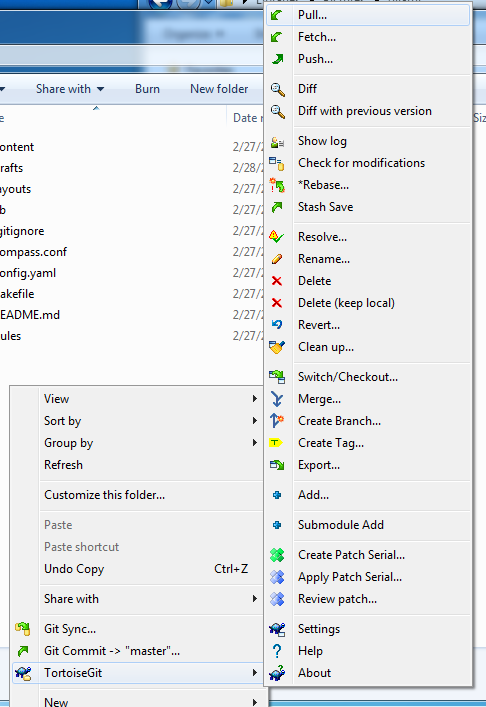
Throw in a title and a description for the pull request, and then hit Submit. We will be sent emails and will have a chance to add comments to your changes.

An important note is that any commits you send to your fork before the pull request is accepted will be merged into that pull request. This lets you make any changes in response to comments, but can be a bit of a surprise if you don't know about it ahead of time.

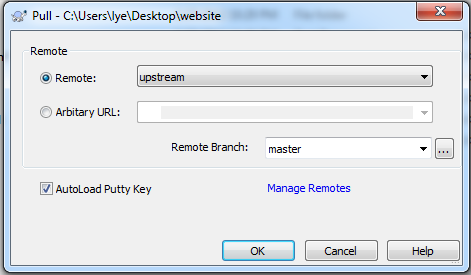
When the pull request is accepted, you'll be notified by email.

**Getting upstream changes back into your copy**

As all of this is going on, we'll be making changes and sending pull requests. To migrate those changes into your codebase, you'll need to use a Pull command. From the right-click context menu, select Pull.



In the dialog that pops up, select upstream as the remote.



Then just hit OK and any changes made by other people will be pulled into your local repository. They'll be sent to your Github fork when you next push to it.

**Conclusion**

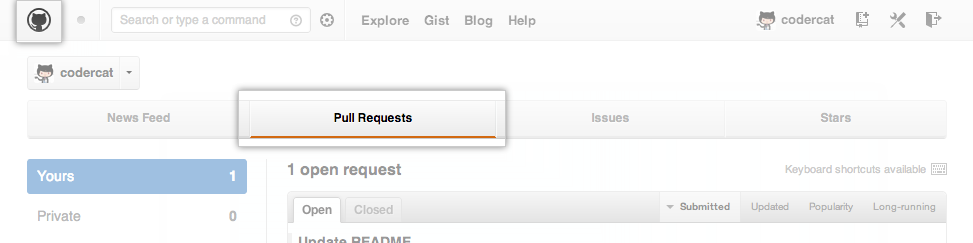
In this way, TortoiseGit and Github are connected. There are built-in tools to establish a proper code-review workflow.

**Accepting Pull Requests and Merging changes to upstream repo**

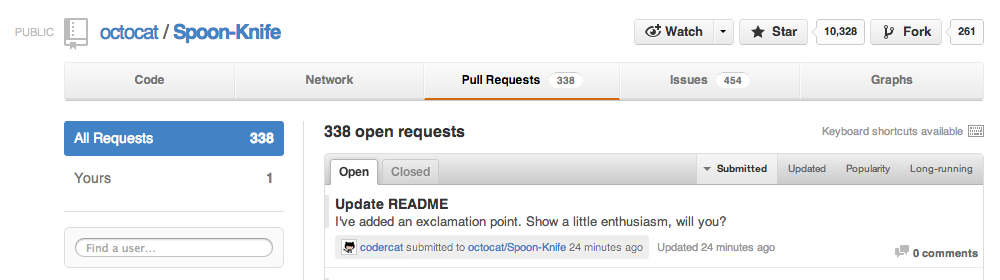
**By [temcocontrols/T3000\_Building\_Automation\_System]**

**Managing Pull Requests**

All pull requests sent or received by you are browsable through the pull request dashboard.



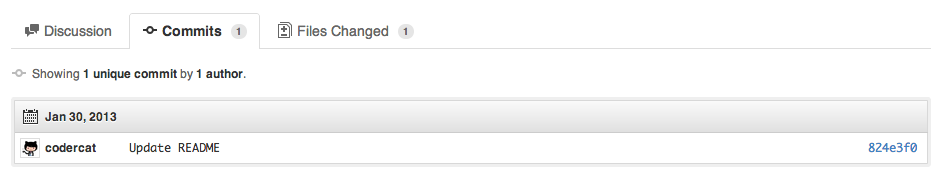
Pull requests for a specific repository are also browsable by anyone with access by visiting the *Pull Requests* page.



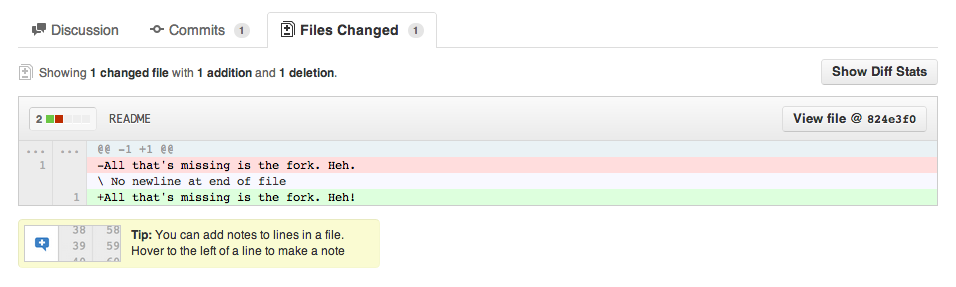
The pull request dashboard and the repository pull request list support a wide range of filtering and sorting controls. Use them to narrow down the list to the pull requests you're interested in.

**Reviewing Proposed Changes**

When you receive a pull request, the first thing to do is review the set of proposed changes. Pull requests are tightly integrated with the underlying git repository, so you can see exactly what commits would be merged should the request be accepted:

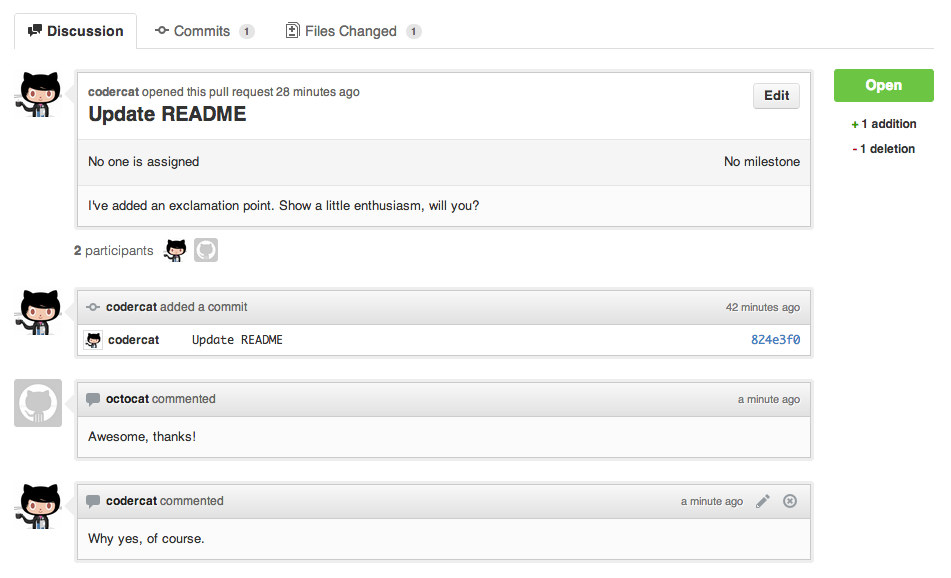


You can also review the cumulative diff of all file changes across all commits.



**Pull Request Discussion**

After reviewing the basic description, commits, and cumulative diff, the person tasked with applying the changes may have questions or comments. Perhaps the coding style doesn't match project guideline, or the change is missing unit tests, or maybe everything looks great and some props are in order. The discussion view is designed to encourage and capture this type of discussion.



The discussion view starts with the pull request's original title and description and then captures additional activity to display chronologically from there. Any of the following types of activity are captured as they happen:

* Comments left on the pull request itself.
* Additional commits pushed to the pull request's branch.
* File and line notes left on any of the commits included in the pull request's range.

Pull request comments are Markdown compatible, so you can embed images, use preformatted text blocks, and other formatting supported by Markdown.

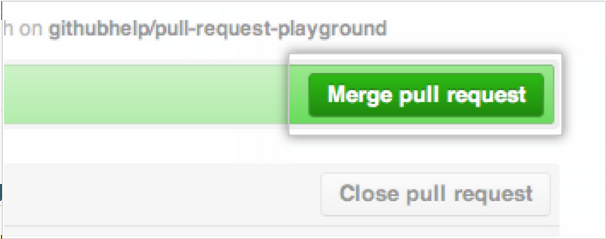
**Merging a Pull Request**

Once a pull request is deemed satisfactory, anyone with push access to the destination repository can perform the merge. There are a variety of ways to accomplish this. Two popular methods are described below.

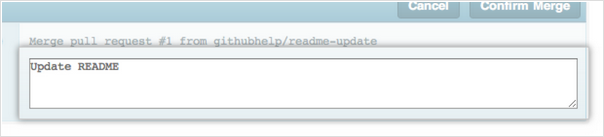
**Merge directly on Github**

If the merge will not have any conflicts, you can merge the pull request online without ever typing a single git command.

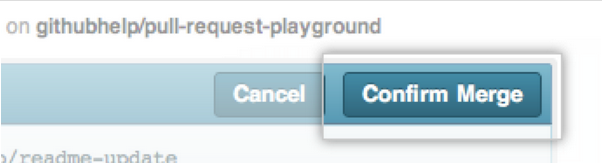
1. Open the pull request's page
2. Click the "Merge pull request" button



1. Enter a commit message



1. Click “Confirm Merge”



# TortoiseGit and Github workflow for T3000\_Building\_Automation\_System project

Before reading this chapter, make sure you have read and understood chapter 6. This chapter only builds on the concepts explained in chapter 6 and makes use of those concepts w.r.t. T3000\_Building\_Automation\_System project. This chapter would be mostly containing screen shots of T3000 project and related steps.

**Github Terms:**

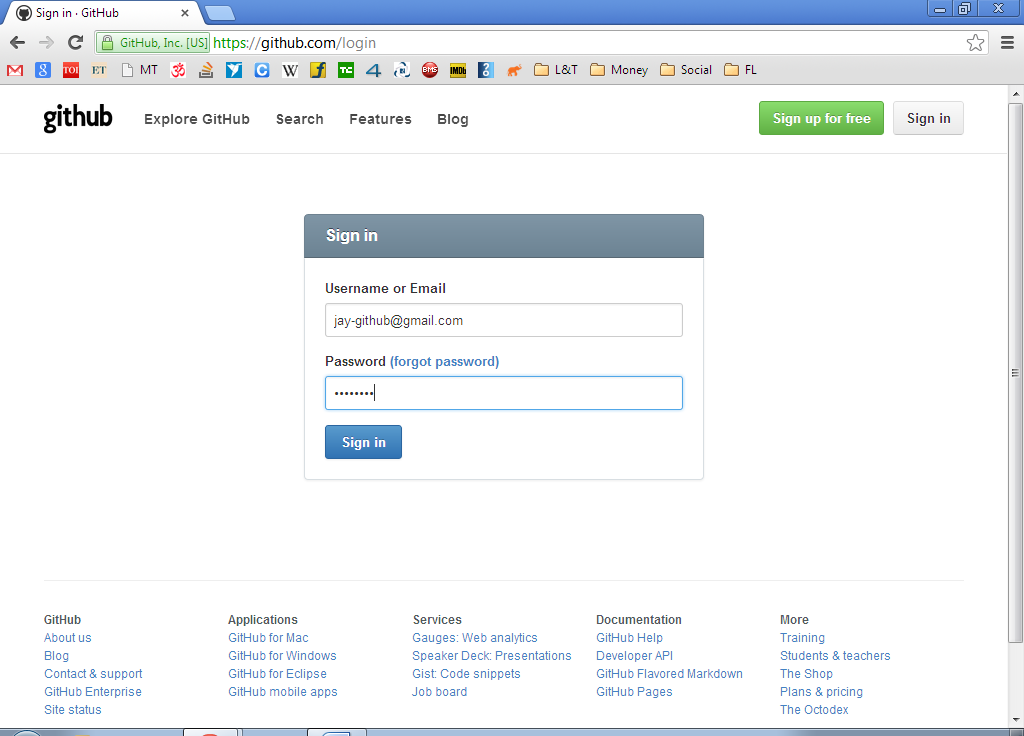
1. **Base repo:** temcocontrols/T3000\_Building\_Automation\_System. Base repo is the main repository of the code. This shall correspond to the finalized work of the team.
2. **Fork repo:** Each fork of the main repo corresponds to a team member’s work. In my case, this repo is: jay-github/ T3000\_Building\_Automation\_System
3. **Branch:** Each branch within the fork and/or in the main repo can correspond to several kinds of things depending on how you want to work. Usually, each branch refers to a version of the project but can also correspond to different channels of development, like hotfixes or experimental work.
4. **Pull Request:** The **pull request** (in the Github ecosystem) corresponds to the task. Every time I want to contribute an isolated finished task to the main repo, I create a pull request corresponding to the **commits** made in that task. These commits are pulled from my **fork** to the **main repo.**

The ecosystem for Temco Controls software development is proposed as follows:

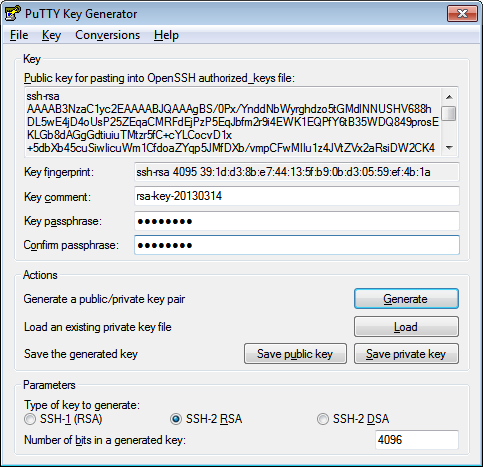
1. temcocontrols/T3000\_Building\_Automation\_System remains the base repo controlled from a single point.
2. Each developer will fork his or her own repo. With this logic, my repository would be jay-github/ T3000\_Building\_Automation\_System.
3. After forking the repo, each developer shall clone that repo in his own local PC. Work there and Push the changes back to his fork.
4. Once the changes are pushed to fork, developer shall send a Pull Request to temcocontrols on github.
5. The single person controlling temcocontrols repo would be able to review the work done by developer and also able to accept and merge or reject the pull request.

Let’s see how this can be done.

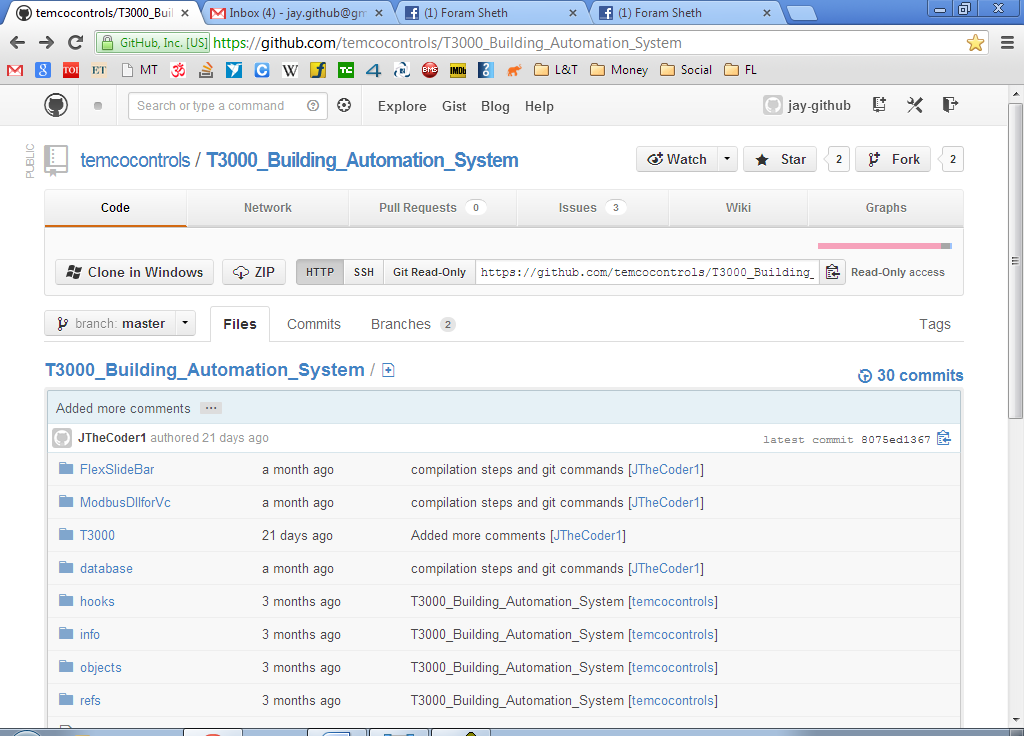
1. Sign in to the developer account.



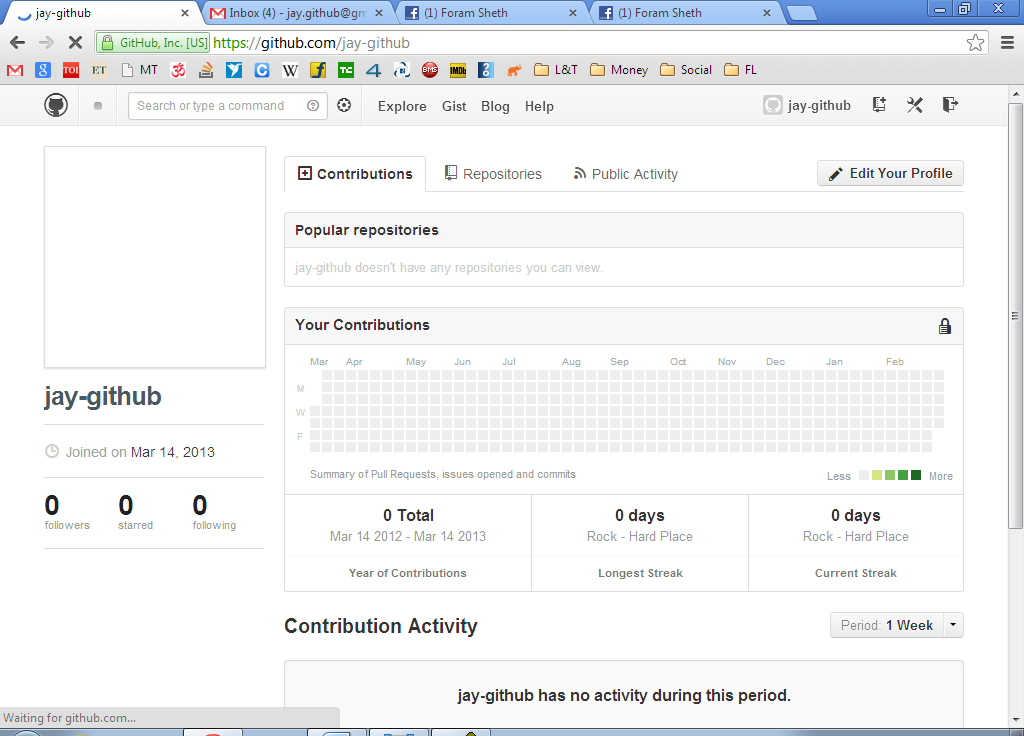
1. Generate SSH key.



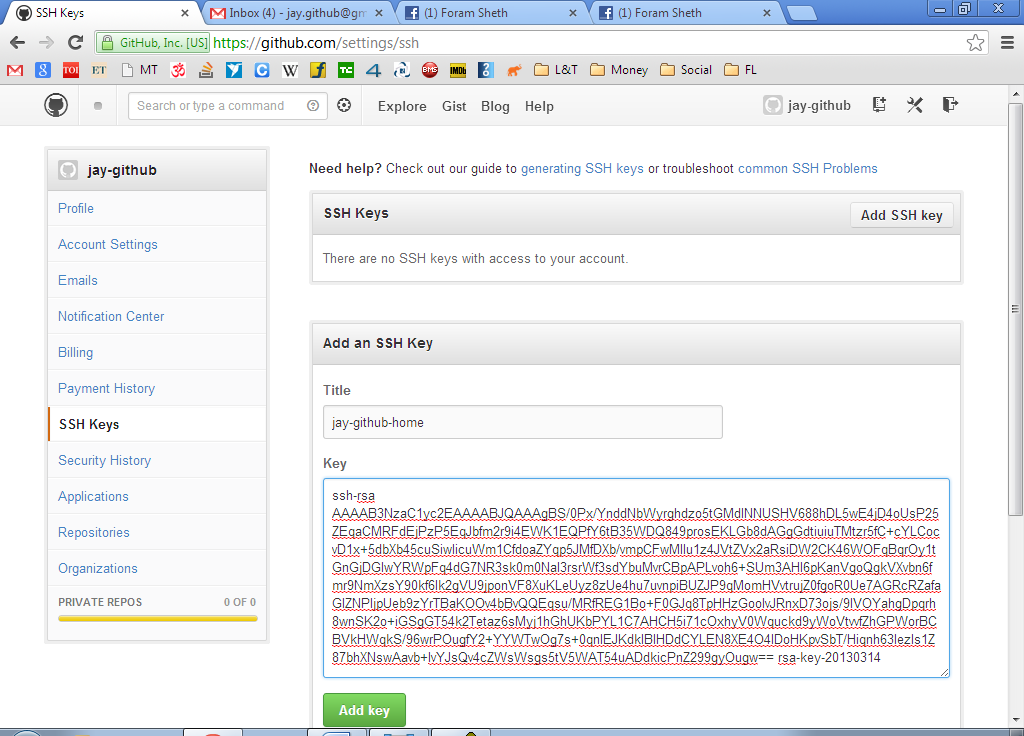
1. Screen below shows signed in home page for developer.



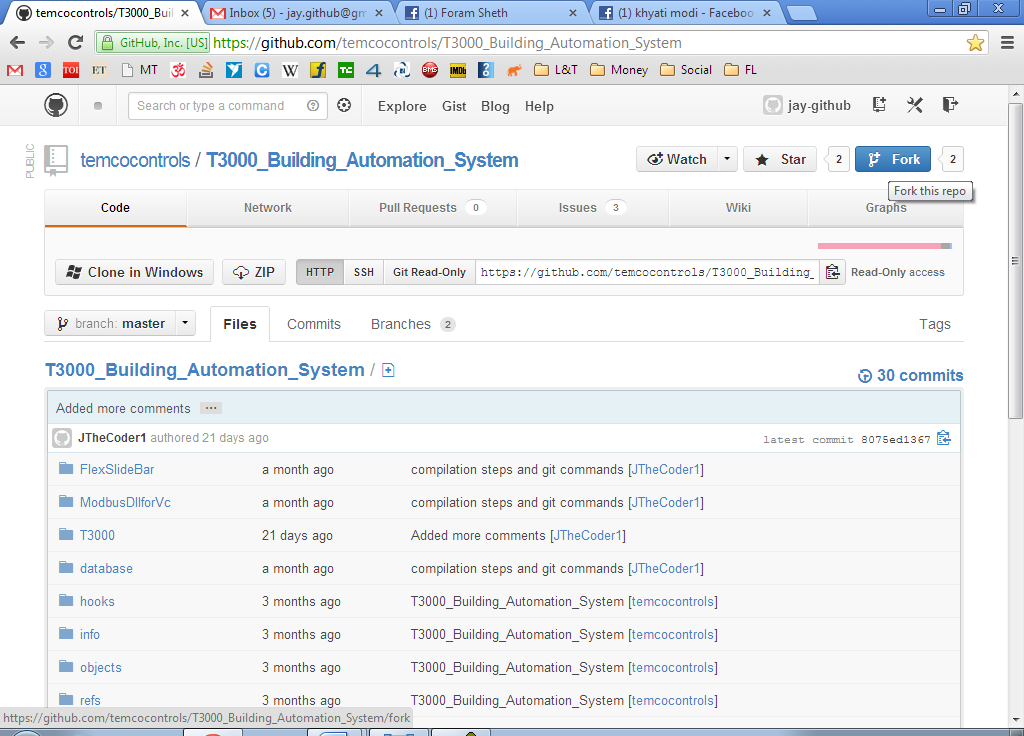
1. Go to Edit Your Profile.



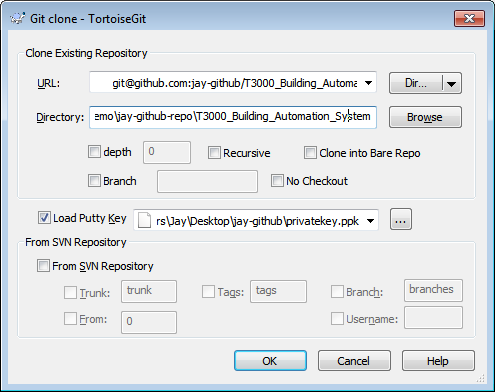
1. Add SSH key.



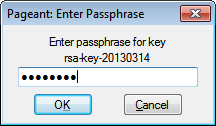
1. Fork temcocontrols/T3000… project



1. Clone fork into local PC using TortoiseGit.



1. Enter passphrase.



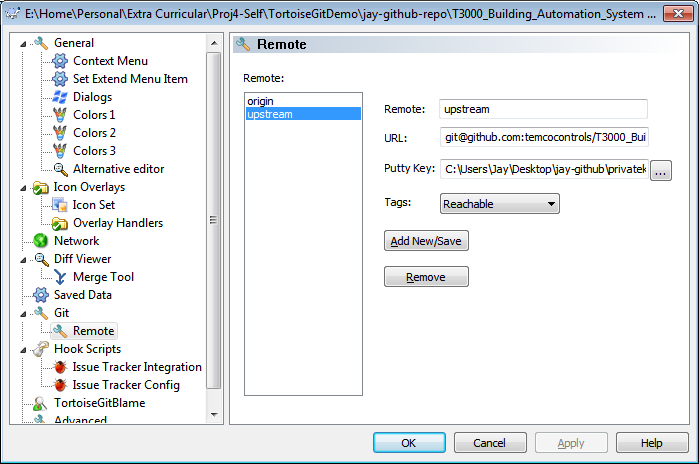
1. Cloning may take some time.



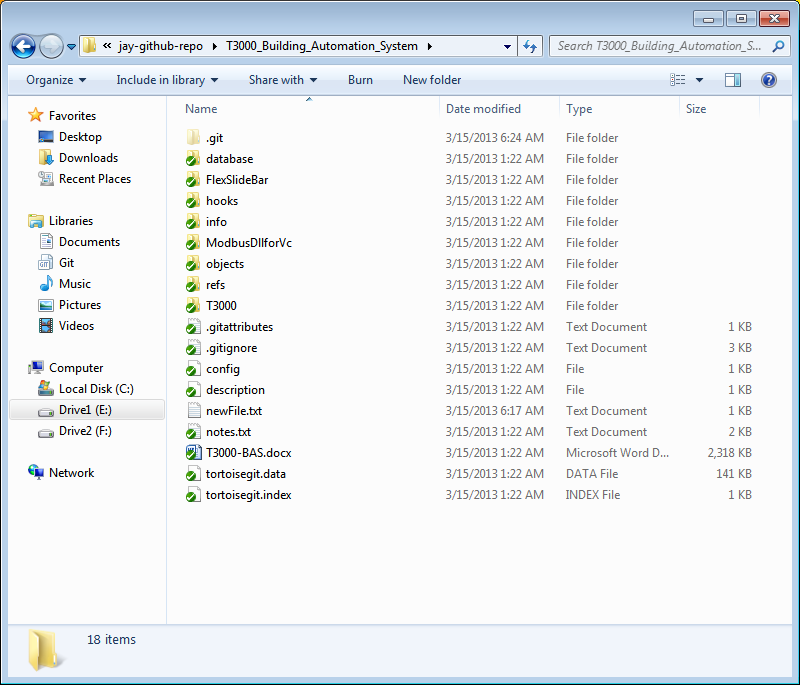
1. Eventually it will get over.



1. Enter details of main repo (temcocontrols/T3000… repo)

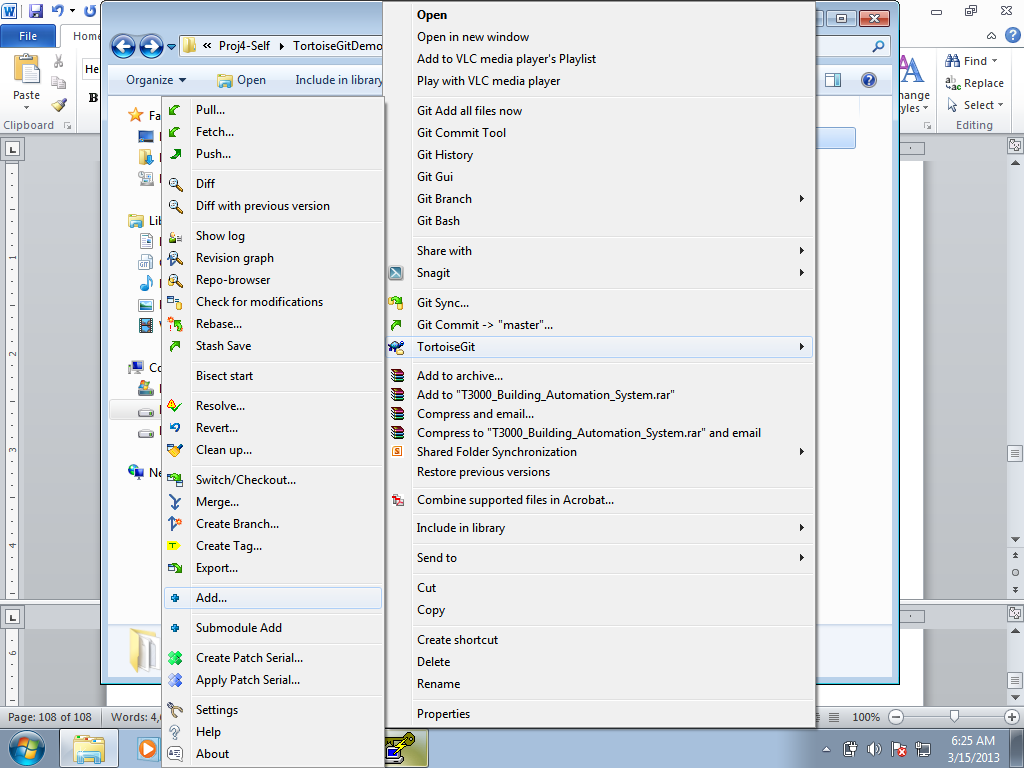


1. Here, for demo purpose, I have added newFile.txt in the repo directory on my local PC.

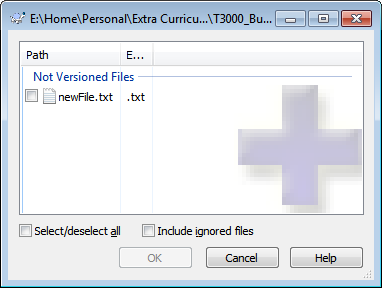


1. Now, it’s time to send my changes to single person controlling temcocontrols repo. This is done by first committing changes, pushing them to own fork and then sending a pull request to temcocontrols.

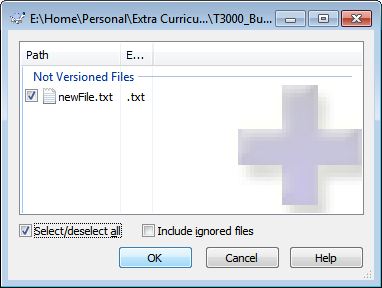
Add… is shown below.



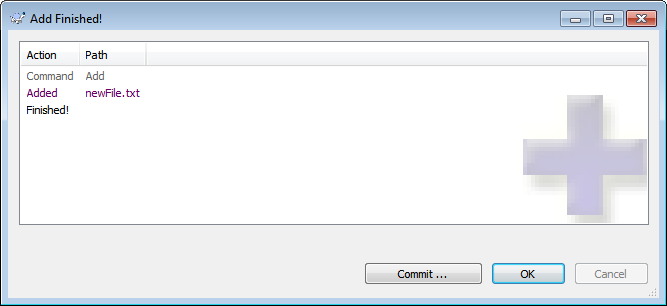
1. TortoiseGit confirms what needs to be added.



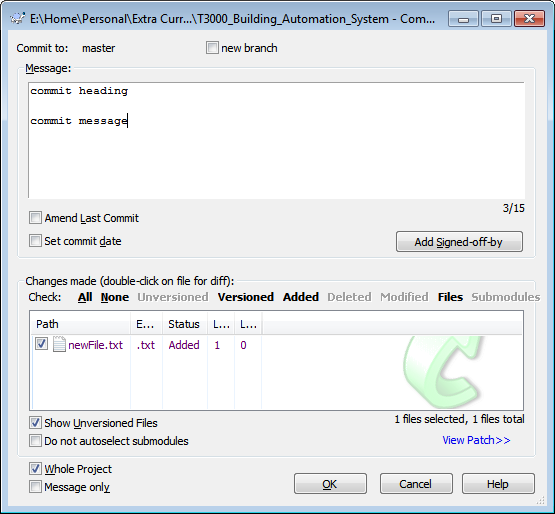
1. Select files to be added and click ok.



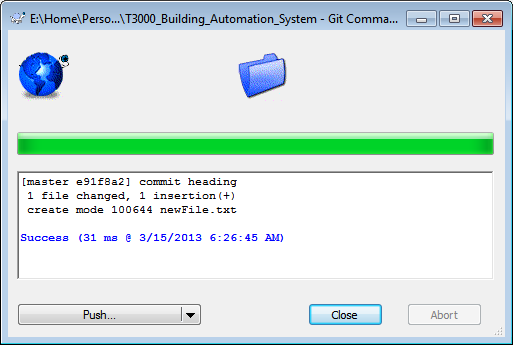
1. File is added and click on Commit… to commit changes on local PC.



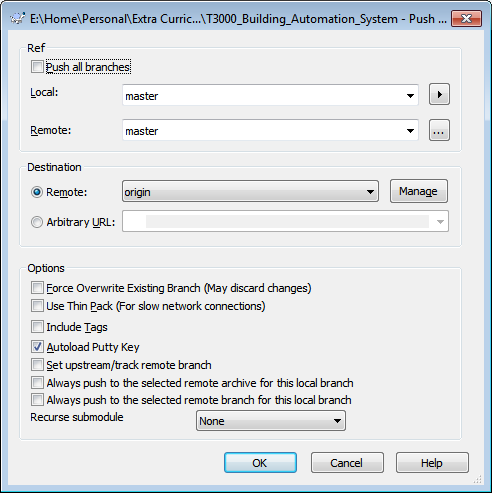
1. Commit dialog box appears. Enter commit message and click ok.



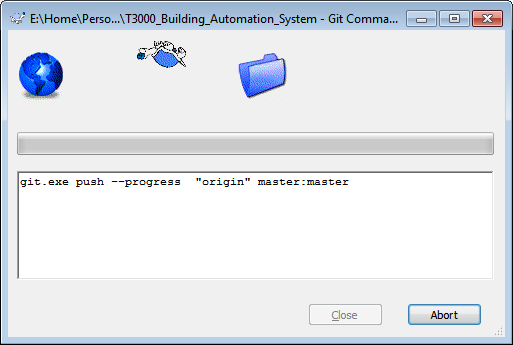
1. Once committed, if you want to send changes to your fork on github, click Push… or click Close.



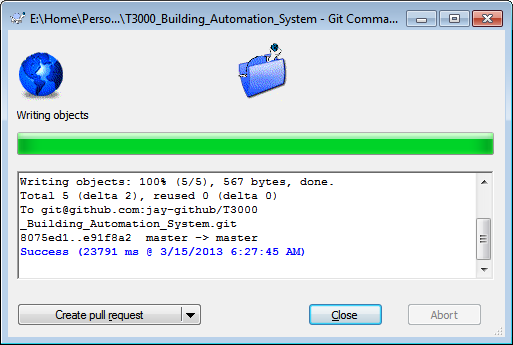
1. Select Destination as origin. Not upstream. Else changes would go to main repo instead of your fork and click ok.



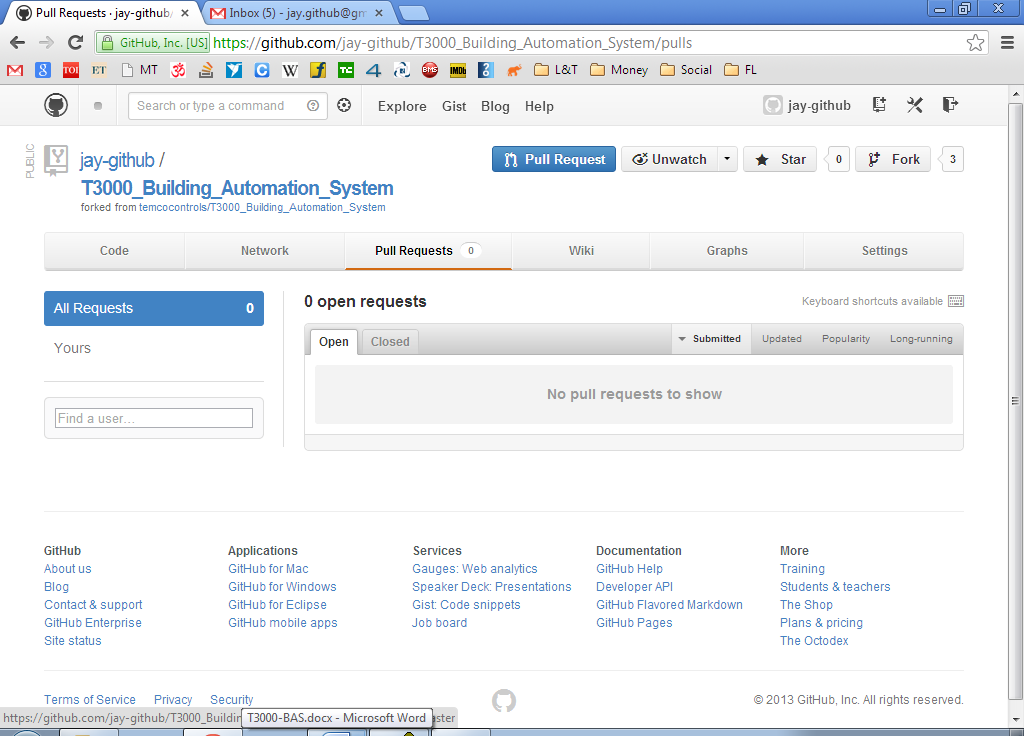
1. Some waiting…



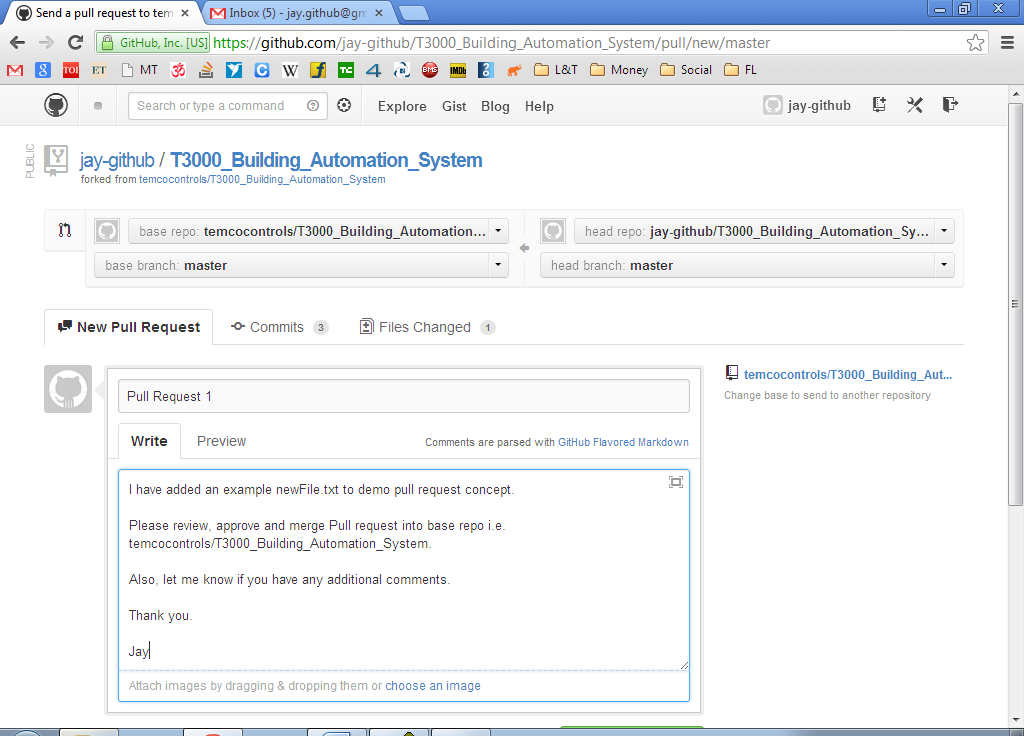
1. And pushing is done.



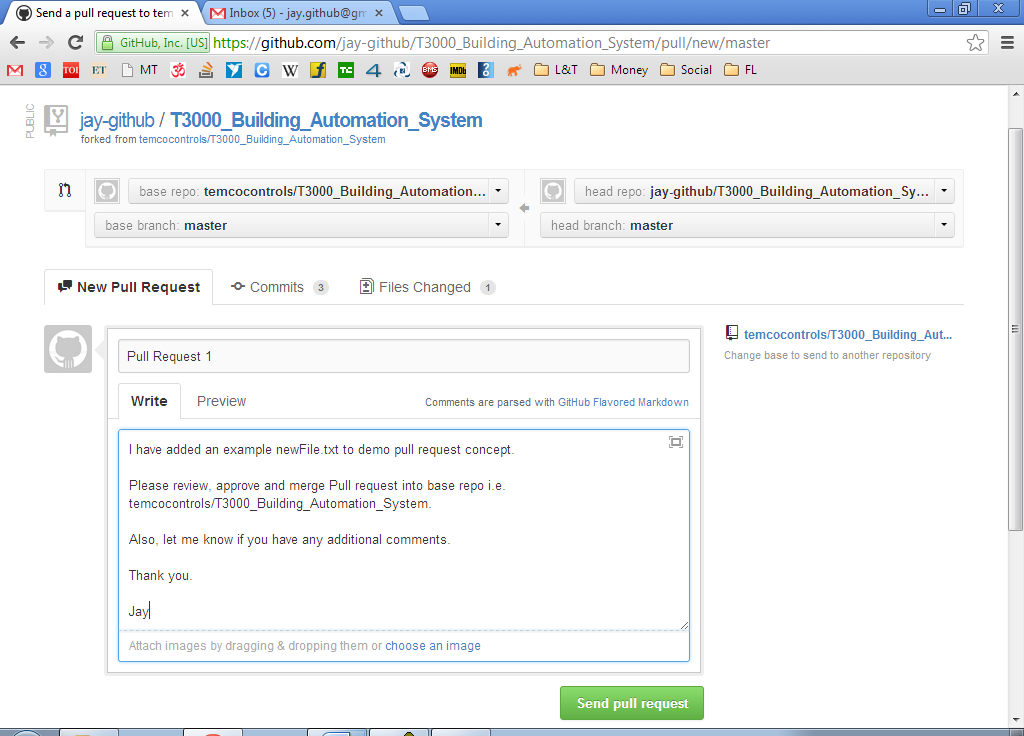
1. Now notify temcontrols of the changes by clicking Pull Request on github website.



1. Enter a title and a message.

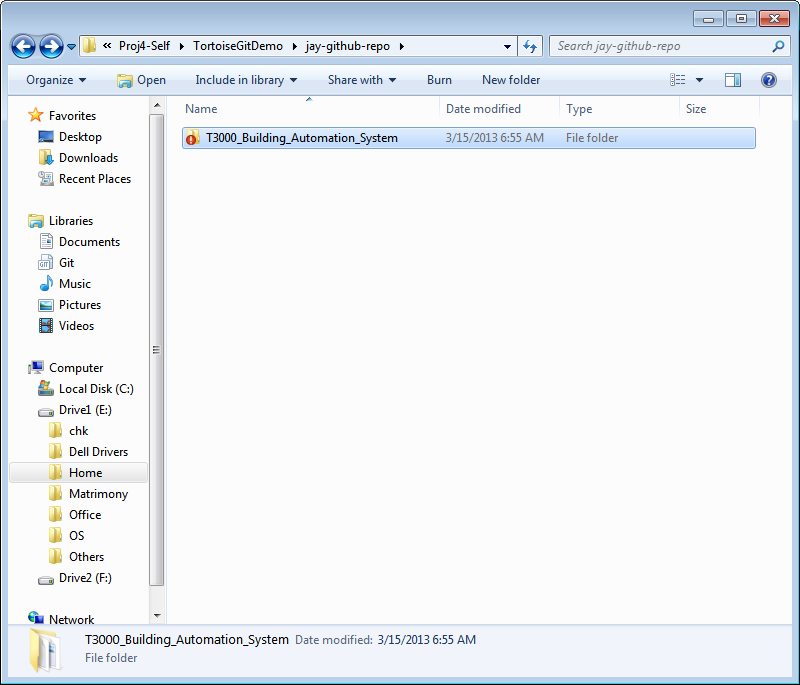


1. Click on Send Pull Request.



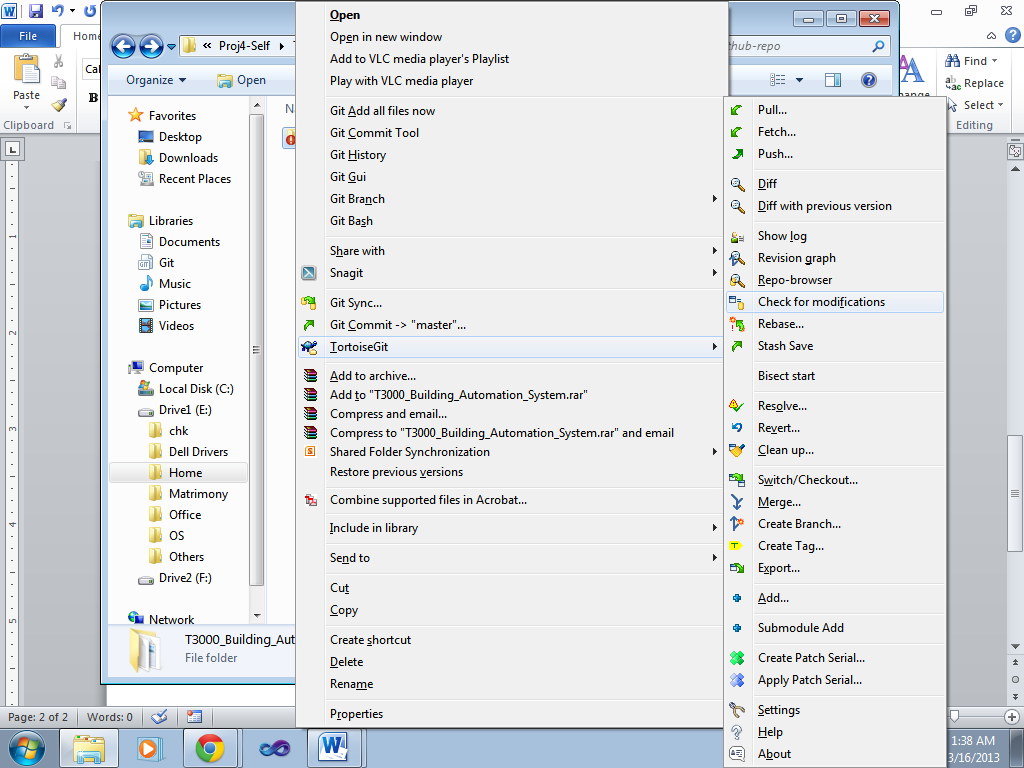
1. Now, single person controlling main repo shall be able to merge the pull request as described in last section of chapter 6.

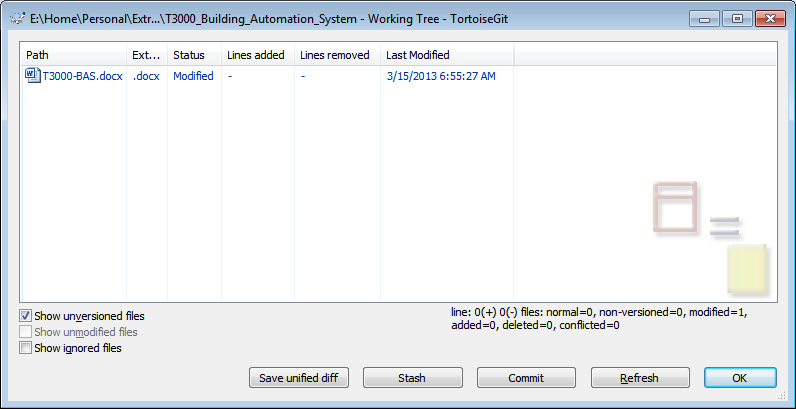
Following are screen shots of adding modifications. (In case Add… does not work)



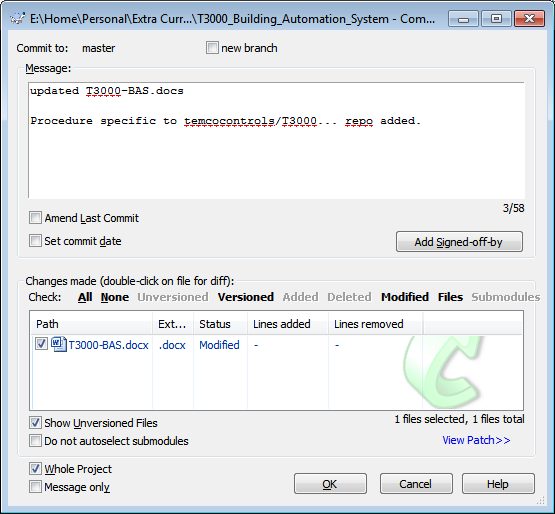


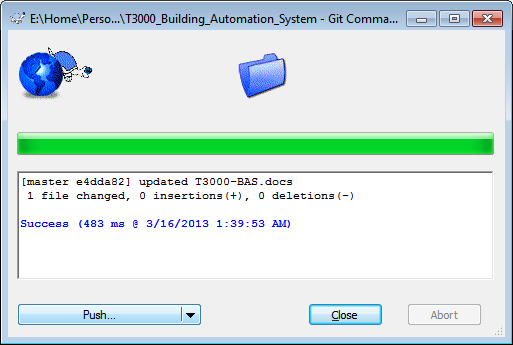
Click on TortoiseGit -> Check for modifications

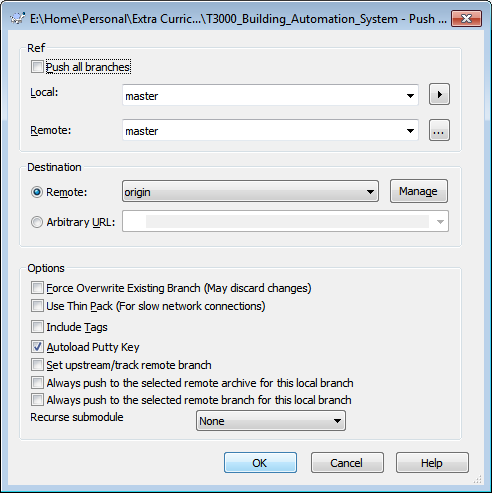


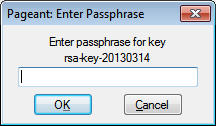


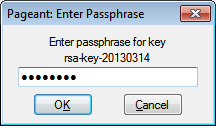
Click on Commit.

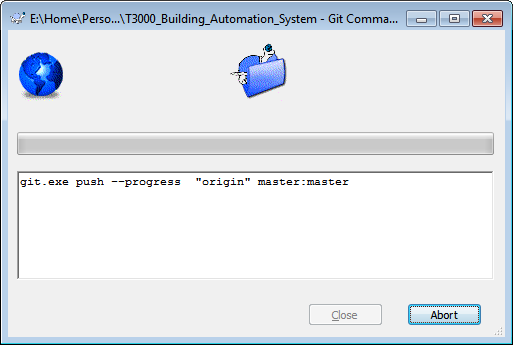


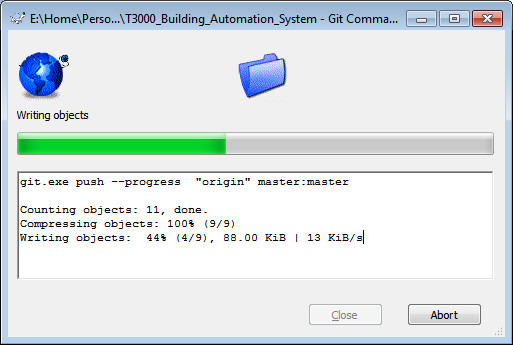


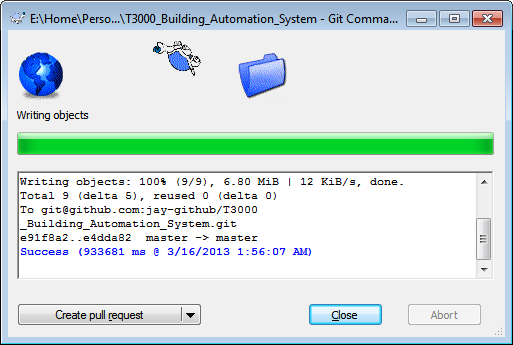












Push to github. TortoiseGit -> Push…

